RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data

Anastasia Dimou
anastasia.dimou@ugent.be

Miel Vander Sande
miel.vandersande@ugent.be

Pieter Colpaert
pieter.colpaert@ugent.be

Ruben Verborgh
ruben.verborgh@ugent.be

Erik Mannens
erik.mannens@ugent.be

Rik Van de Walle
rik.vandewalle@ugent.be

Ghent University – iMinds – Multimedia Lab
Ghent, Belgium

ABSTRACT

Despite the significant number of existing tools, incorporating data from multiple sources and different formats into the Linked Open Data cloud remains complicated. No mapping formalisation exists to define how to map such heterogeneous sources into RDF in an integrated and interoperable fashion. This paper introduces the RML mapping language, a generic language based on an extension over R2RML, the W3C standard for mapping relational databases into RDF. Broadening R2RML's scope, the language becomes source-agnostic and extensible, while facilitating the definition of mappings of multiple heterogeneous sources. This leads to higher integrity within datasets and richer interlinking among resources.

1. INTRODUCTION

Deploying the five stars of the Linked Open Data schema[1] is the *de-facto* way of mapping data. In real-world situations, multiple sources of different formats are part of multiple domains, which in their turn are formed by multiple sources and the relations between them. Approaching the stars as a set of consecutive steps and applying them to a single source every time—as most solutions tend to do—is not always an optimal solution. When mapping heterogeneous data into RDF, such approaches often fail to reach the final goal of publishing *interlinked* data. The semantic representation of each mapped resource is defined independently, disregarding its possible prior definitions and its links to other resources. Manual alignment to their prior appearances is performed by redefining their semantic representations, while links to other resources are defined *after* the data are mapped and published. Nonetheless, as datasets are often shaped gradually, a demand emerges for a well-considered policy regarding mapping and primary interlinking of data in the context of a certain knowledge domains.

For instance, governments publish their data as Open Data and turn them into Linked Open Data *afterwards*. Much of this data, as expected when dealing with many sources, complements each other in the description of different knowledge domain. Therefore, the same concepts appear in multiple data sets, and problematically, often with different identifiers or even in different formats. Furthermore, data is mapped progressively, thus it is important that data publishers incorporate their data in what is already published. Reusing the same unique identifiers for concepts is necessary to achieve this, but it is only possible if prior existing definitions in the same dataset are discovered and if they can be replicated. Otherwise, duplicates will inevitably appear—even within a publisher's own datasets. Identifying, replicating, and keeping those definitions aligned is complicated and the situation aggravates the more data is mapped and published.

Solving this problem requires a uniform, modular, interoperable and extensible technology that supports this need for gradually incrementing datasets. Such a solution can deal with the mapping and primary interlinking of the data, which should take place in a tightly coordinated way instead of as two separate, consecutive actions. This ensures semantic representations of higher quality and datasets with better integrity. To this end, we propose RML, a generic mapping language defined as an extension of R2RML[2], the W3C recommendation for mapping data in relational databases into RDF.

The remainder of the paper is organized as follows: Section 2 discusses related solutions existing today. Section 3 analyzes the requirements of a mapping language, and Section 4 introduces the proposed approach. Next, Section 5 addresses the challenges of implementing an RML processor. Finally, Section 6 outlines our conclusions and future work.

2. RELATED WORK

Several solutions exist to execute mappings from different file structures and serialisations to RDF. For relational databases, different mapping languages beyond R2RML are defined [3] and several implementations already exist[3]. Similarly, mapping languages were defined to support conversion from data in CSV and spreadsheets to the RDF data model. They include the XLWrap's mapping language [5] that converts data in various spreadsheets to RDF, the declarative OWL-centric mapping language Mapping Master's M2 [6] that converts data from spreadsheets into the Web Ontology Language (OWL), Tarql[4] that follows a querying ap-

[1] http://5stardata.info/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

[2] http://www.w3.org/TR/r2rml

[3] http://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations

[4] https://github.com/cygri/tarql

proach and Vertere[5]. The main drawback in the case of most CSV/spreadsheet-to-RDF mapping solutions is the assumption that each row describes an entity (*entity-per-row assumption*) and that each column represents a property.

A larger variety of solutions exist to map from XML to RDF, but to the best of our knowledge, no specific languages were defined for this, apart from GRDDL[6] that essentially provides the links to the algorithms (typically represented in XSLT) that map the data to RDF. Instead, tools mostly rely on existing XML solutions, such as XSLT (e.g., Krextor [4] and AstroGrid-D[7]), XPath (e.g., Tripliser[8]), and XQuery (e.g., XSPARQL [1]). In general, most existing tools deploy mappings from a certain source format to RDF (*per-source approaches*). Few tools provide mappings from *different* source formats to RDF; and those tools actually employ separate source-centric approaches for each of the formats they support. Datalift [7], The DataTank [9], OpenRefine[10], RDFizers [11] and Virtuoso Sponger[12] are the most well-known.

3. MAPPINGS METHODOLOGY

After outlining the limitations of existing solutions, we present the factors that can improve the mappings to produce better integrated datasets and early interlinked resources.

3.1 Limitations of current mapping methods

We identified the following limitations that prevent current practices from achieving well integrated datasets.

Mapping of data on a per-source basis. Most of the current solutions work on a per-source basis: only one source is mapped at once, as opposed to mapping different related sources together, despite covering the same domains or sharing the same formats. As a result, data publishers can only generate resources and links between data appearing within a single source. Their mapping definitions need to be aligned manually when the same resources already appear in the targeting dataset. Thus, data publishers need to redefine and replicate the patterns for the resources' URIs definition every time they appear in a new mapping rule. Furthermore, this is not always possible, as the data included in the one source may not be sufficient to replicate the same URIs. This results in distinct URIs for identical resources, which leads to duplicates within a publisher's own dataset. In addition, the interlinking of the resources generated from different sources has to be performed afterwards.

Mapping data on a per-format basis. Besides the *per-source* approach, most of the current solutions provide a *per-format* approach: only mappings from a certain source format (e.g., XML) are supported. In practice, data publishers need to map various source formats to RDF. Therefore, they need to install, learn, use and maintain different tools for each case separately, which hampers their effort to ensure the integrity of their datasets even more. Alternatively, some end up implementing their own *case-specific* solutions.

Mapping definitions' reusability. The mapping definitions of current solutions are not reusable, as there is no standard formalisation for any source format apart from relational databases, i.e., R2RML. In most cases, the mapping rules are not interoperable as they are tied to the implementation, which prevents their extraction and reuse across different implementations. Moreover, this prohibits reuse of the same mapping rules to map data that describe the same model, but is serialized in different initial formats.

3.2 Requirements for generic mappings

To achieve datasets with better integrated and richer interlinked resources, the aforementioned issues should be addressed during the mapping phase, rather than later. A set of factors that contribute to this are outlined below.

Uniform and interoperable mapping definitions. Since we require a uniform way of dealing with different source serializations, the mapping definitions should be defined independently of the references to the input data. The same mappings may then be reused across different sources—as long as they capture the same context (i.e., the same RDF representations)—only by changing the reference to the input source that holds the information. For example, a *performance* described in a JSON file and an *exhibition* described in an XML file may take place at the same location, indicated by an identical longitude/latitude pair. We only need a single mapping definition to describe their location, adjusted to point to respectively the JSON objects and the XML elements that hold the corresponding values. Therefore, we require a *modular language* in which the references to the data extracts and the mapping definitions are distinct and not interdependent. Thereby, the mapping definitions can be reused across different implementations for different source formats, reducing the implementation and learning costs.

Robust cross-references and interlinking. Redefining and replicating patterns every time a new input source is integrated should be avoided. Publishers should be able to uniquely define the pattern that generates a resource and refer to its definition every other time this resource is mapped (in this way enriched), which has the following three advantages: First, possible modifications to the patterns, or data values appearing in the patterns that generate the URIs, are propagated to every other reference of the resource, making the interlinking more robust. Second, taking advantage of this integrated solution, *cross-references* among sources become possible; links between resources in different input sources are defined already on mapping level. Third, and most significant, when data publishers want to map a new source, their new mappings are defined taking advantage of and automatically aligning to the existing ones.

Extending the aforementioned example, the *venue* where the *performance* and the *event* take place is the same. When the input source for the *performances* was mapped, the mappings for the possible *venues* were defined considering certain identifiers to define their URIs. Once the *exhibitions* are about to be mapped, the data publisher might not be able to reuse the existing mapping definition for the *venues* as the identifiers are not included in the dataset to replicate

[5] https://github.com/knudmoeller/Vertere-RDF
[6] http://www.w3.org/TR/grddl/
[7] http://www.gac-grid.de/project-products/Software/XML2RDF.html
[8] http://daverog.github.io/tripliser/
[9] http://thedatatank.com
[10] http://openrefine.org/
[11] http://simile.mit.edu/wiki/RDFizers
[12] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger

the same patterns. However, the *venue name* might be considered to determine the binding. Then, the existing mapping definition can be referred to generate the same URIs and, thus enrich the existing resource with new attributes and interlink data from the newly mapped dataset to the existing one. As the original input source is an Open Data set that can be referenced, it is always available to be used to support the mapping of the new data. Summarizing, the definition of the links between resources in different sources—even if they are in different formats—happens on the mapping level instead of during a subsequent interlinking step.

Scalable mapping language. As the references to the data extracts and the mapping definitions are distinct and not interdependent, the pointer to the input source's data can be adjusted to each case. Such modular solution leads to correspondingly modular implementations that perform the mappings in a uniform way, independent of the input source. They only adjust the respective extraction mechanism depending on the input source. *Case-specific* solutions exist because complete generic solutions fail, as it is impossible to predict every potential input. A scalable solution addresses what can be defined in a generic way for all possible different input sources and scales over what cannot. In order to support emerging needs, it should allow extensions with source-specific references, addressed on a case-specific level.

4. RML MAPPING LANGUAGE

The *RDF Mapping language* (RML) is a generic mapping language defined to express customized mapping rules from heterogeneous data structures and serializations to the RDF data model. RML is defined as a superset of the W3C-standardized mapping language R2RML, aiming to extend its applicability and broaden its scope.

4.1 R2RML

R2RML is defined to express customized mappings only from data in relational databases to datasets represented using the RDF data model. In R2RML, the mapping to the RDF data model is based on one or more Triples Maps and occur over a Logical Table iterating on a *per-row* basis. A Triples Map consists of three main parts: the Logical Table (`rr:LogicalTable`), the Subject Map and zero or more Predicate-Object Maps. The Subject Map (`rr:SubjectMap`) defines the rule that generates unique identifiers (URIs) for the resources which are mapped and is used as the subject of all the RDF triples that are generated from this Triples Map. A Predicate-Object Map consists of Predicate Maps, which define the rule that generates the triple's predicate and Object Maps or Referencing Object Maps, which defines the rule that generates the triple's object. 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* (`rr:constant`) that always generates the same RDF term, or a *column-valued term map* (`rr:column`) that is the data value of a referenced column in a given Logical Table's row, or a *template-valued term map* (`rr:template`) that is a valid string template that can contain referenced columns.

Furthermore, R2RML supports cross-references between Triples Maps, when the subject of a Triples Map is the same as the object generated by a Predicate-Object Map. A Referencing Object Map (`rr:RefObjectMap`) is used then to point to the Triples Map that generates on its Subject Map the corresponding re-

	R2RML	RML
Input Reference	Table Name	Source
Value Reference	Column	Reference
Iteration model	per row(implicit)	defined
Source Expression	SQL (implicit)	Reference Formulation

Table 1: R2RML Vs RML.

source, the so-called Referencing Object Map's Parent Triples Map. If the Triples Maps refer to different Logical Tables, a join between the Logical Tables is required. The join condition (`rr:joinCondition`) performs the join exactly as a join is executed in SQL. The join condition consists of a reference to a column name that exists in the Logical Table of the Triples Map that contains the Referencing Object Map (`rr:child`) and a reference to a column name that exists in the Logical Table of the Referencing Object Map's Parent Triples Map (`rr:parent`).

4.2 RML

RML keeps the mapping definitions as in R2RML but excludes its database-specific references from the core model. The potential broad concepts of R2RML, which were explained previously [2], are formally designated in the frame of the RML mapping language and are elaborated upon here. The primary difference is the potential input that is limited to a certain database in the case of R2RML, while it can be a broad set of (one or more) input sources in the case of RML. Table 1 summarizes overall the RML's extensions over R2RML entailed because of the broader set of possible input sources.

RML provides a generic way of defining the mappings that is easily transferable to cover references to other data structures, combined with case-specific extensions, but always remains backward compatible with R2RML as relational databases form such a specific case. RML considers that the mappings to RDF of sets of sources that all together describe a certain domain, can be defined in a combined and uniform way, while the mapping definitions may be re-used across different sources that describe the same domain to incrementally form well-integrated datasets, as displayed at Figure 1.

An RML mapping definition follows the same syntax as R2RML. The RML vocabulary namespace is `http://semweb.mmlab.be/ns/rml#` and the preferred prefix is *rml*. More details about the RML mapping language can be found at `http://semweb.mmlab.be/rml`. Defining and executing a mapping with RML requires the user to provide a valid and well-formatted *input dataset* to be mapped and the mapping definition (*mapping document*) according to which the mapping will be executed to generate the data's representation using the RDF data model (*output dataset*). Data cleansing is out of the scope of the language's definition and, if necessary, should be performed in advance. An extract of two heterogeneous input sources is displayed at Listing 1, an example of a corresponding mapping definition is displayed at Listing 3 and the produced output at Listing 2.

Logical Source. A Logical Source (`rml:LogicalSource`) extends R2RML's Logical Table and is used to determine the input source with the data to be mapped. The R2RML Logical Table

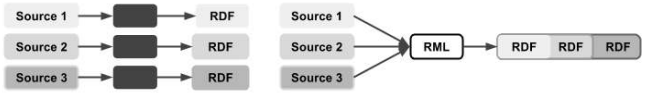

Figure 1: Mapping sources without and with RML

```
{ ...  "Performance" :
         { "Perf_ID": "567",
           "Venue": {  "Name": "STAM",
                       "Venue_ID": "78" },
           "Location": { "long": "3.717222",
                         "lat": "51.043611" } } , ... }
<Events> ...
  <Exhibition id="398">
    <Venue> STAM </Venue>
    <Location>
      <lat>51.043611</lat>
      <long>3.717222</long>
    </Location>
  </Exhibition> ... ...
</Events>
```

Listing 1: `performances.json` and `exhibitions.xml`

```
ex:567      ex:venue    ex:78 ;
            ex:location ex:3.717222,51.043611 .
ex:398      ex:venue    ex:78 ;
            ex:location ex:3.717222,51.043611 .
ex:3.717222,51.043611   ex:lat      ex:3.717222
                        ex:long     ex:51.043611.
```

Listing 2: The expected output.

definition determines a database's table, using the Table Name (`rr:tableName`). In the case of RML, a broader reference to any input source is required. Thus, the Logical Source and source (`rml:source`) are introduced respectively to specify the input.

Reference Formulation. RML needs to deal with different data serialisations which use different ways to refer to their elements/objects. But, as RML aims to be generic, not a uniform way of referring to the data's elements/objects is defined. R2RML uses columns' names for this purpose. In the same context, RML considers that any reference to the Logical Source should be defined in a form relevant to the input data, e.g. XPath for XML files or JSONPath for JSON files. To this end, the Reference Formulation (`rml:referenceFormulation`) declaration is introduced indicating the formulation (for instance, a standard or a query language) used to refer to its data. At the current version of RML, the `ql:CSV`, `ql:XPath` and `ql:JSONPath` Reference Formulations are predefined.

Iterator. While in R2RML it is already known that a *per-row* iteration occurs, as RML remains generic, the iteration pattern, if any, can not always be implicitly assumed, but it needs to be determined. Thereafter, the iterator (`rml:iterator`) is introduced. The iterator determines the iteration pattern over the input source and specifies the extract of the data mapped during each iteration. For example, the `"$.[*]"` determines the iteration over a JSON file that occurs over the object's outer level. The iterator is not required in the case of tabular sources as the default *per-row* iteration is implied or if there is no need to iterate over the input data.

Logical Reference. A *column-valued term map*, according to R2RML, is defined using the property `rr:column` which determines a column's name. In the case of RML, a more generic property is introduced `rml:reference`. Its value must be a valid reference to the data of the input dataset. Therefore, the reference's value should be a valid expression according to the Reference Formulation defined at the Logical Source, as well as the string template used in the definition of a *template-valued term map* and the iterator's value. For instance, the iterator, the subject's *template-valued term map* and the object's *reference-valued term map* are all valid JSONPath expressions.

Referencing Object Map. The last aspect of R2RML that is extended in RML is the Referencing Object Map. The join condition's child reference (`rr:child`) indicates the reference to the data value (using an `rml:reference`) of the Logical Source that contains the Referencing Object Map. The join condition's *child reference* (`rr:parent`) indicates the reference to the data extract (`rr:reference`) of the Referencing Object Map's Parent Triples Map. The reference is specified using the Reference Formulation defined at the current Logical Source. The join condition's *parent reference* indicates the reference to the data extract (`rml:reference`) of the Parent Triples Map. The reference is specified using the Reference Formulation defined at the Parent Triples Map Logical Source definition. Therefore, the *child reference* and the *parent reference* of a join condition may be defined using different Reference Formulations, if the Triples Map refers to sources of different format.

```
1  <#PerformancesMapping>
2   rml:logicalSource [
3    rml:source "http://ex.com/performances.json";
4    rml:referenceFormulation ql:JSONPath;
5    rml:iterator "$.Performance.[*]" ];
6    rr:subjectMap [ rr:template "http://ex.com/{Perf_ID}" ];
7   rr:predicateObjectMap [ rr:predicate ex:venue;
8    rr:objectMap [ rr:parentTriplesMap <#VenueMapping> ] ];
9   rr:predicateObjectMap [ rr:predicate ex:location;
10    rr:objectMap [ rr:parentTriplesMap <#LocationMapping> ] ].
11
12 <#VenueMapping>
13  rml:logicalSource [
14   rml:source "http://ex.com/performances.json";
15   rml:referenceFormulation ql:JSONPath;
16   rml:iterator "$.Performance.Venue.[*]" ];
17  rr:subjectMap [ rr:template "http://ex.com/{Venue_ID}" ].
18
19 <#LocationMapping>
20  rml:logicalSource [ ......... ];
21  rr:subjectMap [ rr:template "http://ex.com/{lat},{long}" ];
22  rr:predicateObjectMap [ rr:predicate ex:long;
23    rr:objectMap [ rml:reference "long" ] ]
24  rr:predicateObjectMap [ rr:predicate ex:lat;
25    rr:objectMap [ rml:reference "lat" ] ] .
26
27 <#ExhibitionMapping>
28  rml:logicalSource [
29   rml:source "http://ex.com/exhibitions.xml";
30   rml:referenceFormulation ql:XPath;
31   rml:iterator "/Events/Exhibition" ];
32  rr:subjectMap [ rr:template "http://ex.com/{@id}" ];
33   rr:predicateObjectMap [ rr:predicate ex:location;
34    rr:objectMap [ rr:parentTriplesMap <#LocationMapping> ] ];
35  rr:predicateObjectMap [ rr:predicate ex:venue;
36    rr:objectMap [ rr:parentTriplesMap <#VenueMapping>;
37     rr:joinCondition [
38     rr:child "$.Performance.Venue.Name";
39     rr:parent "/Events/Exhibition/Venue" ] ] ] .
```

Listing 3: An RML mapping definition.

5. RML PROCESSING

RML is highly extensible towards new source formats, allowing different levels of support. On processing level that adds some complexity as it demands the processor to be scalable to support different input sources, in a uniform way. To deal with these caveats, RML relies on expressions in a *target expression language* relevant to the source format to refer to the values of the sources while uses the RML syntax for the rest of the mapping definition. This *target expression language* needs to be tied to its format and should act as a *point of reference* to the values in a source.

Expressions can be located wherever values need to be extracted from the source (Term maps and `rr:iterator`) and have to be valid according to the formulation specified in

the Triples Map (`rr:referenceFormulation`). In order to deal with these embedded expressions, an RML processor is required to have a modular architecture where the extraction and mapping modules are executed independently of each other. When the RML mappings are processed, the mapping module deals with the mappings' execution as defined at the mapping document in RML syntax, while the extraction module deals with the target language's expressions.

Mapping Models

An RML processor can be implemented using two alternative models: mapping-driven, data-driven or in a hybridic fashion following any combination of the two solutions that turns the processor to better perform.

Mapping-driven. In this model, the processing is driven by the mapping module. The processor processes each Triples Maps in a consecutive order. Based on the defined expression language, each Triples Map is delegated to a language-specific *sub-extractor*. For each Triples Map, its delegated sub-extractor iterates over the source data as the Triples Map's Iterator specifies. For each iteration the mapping module requests an extract of data from the extraction module. The defined Subject Map and Predicate-Object Maps are applied and the corresponding triples are generated. The execution of dependent Triples Maps, because of joins, is triggered by the Parent Triples Map and a *nested* mapping process occurs.

Data-driven. In this model, the processing is driven by the extractor module, namely the data sources. The processor extracts beforehand the iteration patterns, if any, from the Triples Maps. Each defined dataset is integrated by its language-specific sub-extractor. Based on the defined expression language and the iterator, each Triples Map is delegated to a specific *sub-mapper*. For each iteration, a data extract is passed to the processor, which in turn, delegates the extract of data to the corresponding sub-mapper. The defined Subject Map and Predicate-Object Maps are applied and the corresponding triples are generated. The execution of dependent Triples Maps, because of joins, is triggered by the Parent Triples Map and a *nested* mapping-driven process occurs.

The efficiency of the processor can be increased by scheduling the execution of the present expressions in an intelligent way. The mapping-driven model allows the most straightforward implementation, since Triples Maps are processed independently from each other. However, because of this, avoiding multiple passes over the same dataset is difficult. With execution planning, the number of file passes can be reduced to the bare minimum, but can not be one for all cases. The data-driven model does not have this problem, since one element of a single dataset can activate all related mappings. The execution planning does become more complex, since all dependencies have to be resolved beforehand. Note that we deliberately ignore storing files into memory, which would solve the multiple passes for the mapping-driven approach. We only consider a *streaming* solution, since RML can be used to process datasets too big for the processor' s memory. We accept a longer mapping time in trade of lower memory usage. A side-effect of a streaming approach, is the inability to support some features of expression languages. For instance, XPath has look-ahead functionality that requires access to data which is not yet known. Thus, we can only support a subset. Nevertheless, in practice, most of the expressions only require functionality within this subset.

We created a prototype RML processor implementation in Java based on the mapping-driven model which is available at `https://github.com/mmlab/RMLProcessor`.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a novel approach for mapping heterogeneous sources into RDF using the RML, an easily extendable mapping language that significantly reduces the effort for integrated mapping of heterogeneous resources. Our proposed solution efficiently solves the limitations outlined (Section 3.1) by addressing the factors presented (Section 3.2) that could improve the dataset' s integrity and their resources' interlinking, incorporates the data publisher's URI policy in a well considered *mapping policy*. The *per-format* and *per-file* mapping models followed so far get surpassed, leading to contingent data integration and interlinking at a primary stage. The language's extensibility is self-evident as the whole solution relies on the extension of the R2RML mapping language and arose in a progressive way, as it was initially performed to accommodate mappings from the XML format to the RDF data model and later on was re-used as such for mappings of data appearing in JSON.

In the future, a thorough evaluation of RML's efficiency and effectiveness will be performed. Furthermore, RML can be extended to support views on sources, built by queries. This captures, to an extent, the issue of data cleaning and transformation enhancing its applicability. Next, the efficiency of RML processing can be improved. A possible optimization is the use of execution plans that efficiently arrange the execution order depending on their dependencies. Finally, RML could be used to specify the triples' provenance, by taking advantage of the RDF-nature of the mapping documents.

7. REFERENCES

[1] S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres. Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, 1(3):147–185, 2012.

[2] A. Dimou, M. Vander Sande, P. Colpaert, E. Mannens, and R. Van de Walle. Extending R2RML to a Source-independent Mapping Language for RDF. In *International Semantic Web Conference (Posters and Demos)*, 2013.

[3] M. Hert, G. Reif, and H. C. Gall. A comparison of RDB-to-RDF mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 25–32. ACM, 2011.

[4] C. Lange. Krextor - an extensible framework for contributing content math to the Web of Data. In *Proceedings of the 18th Calculemus and 10th international conference on Intelligent computer mathematics*, MKM'11, pages 304–306. Springer-Verlag, 2011.

[5] A. Langegger and W. Wöß. XLWrap – Querying and Integrating Arbitrary Spreadsheets with SPARQL. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 359–374. Springer-Verlag, 2009.

[6] M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. Mapping Master: a flexible approach for mapping spreadsheets to OWL. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part II*, ISWC'10, pages 194–208. Springer-Verlag, 2010.

[7] F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Képéklian, F. Cotton, J. Euzenat, Z. Fan, P.-Y. Vandenbussche, and B. Vatant. Enabling Linked Data publication with the Datalift platform. In *Proc. AAAI workshop on semantic cities*, 2012.