A ShExML Perspective on Mapping Challenges: Already Solved Ones, Language Modifications and Future Required Actions

Herminio García-González

IT and Communications Service, University of Oviedo, Asturias, Spain
garciaherminio@uniovi.es

Abstract. Data mapping languages allow users to create knowledge graphs with lower cost and time. Some challenges cannot be solved with state-of-the-art languages and tools, though. Thus, in this paper we use and modify ShExML to deal with some of them. We see how some challenges were already solved, which modifications we had to perform to cover others, and how the rest of them could be covered in future versions. Then, we establish a demonstration on language integrity after changes and a discussion on performed and upcoming changes. These solutions, alongside the discussion and joint analysis of other languages and tools solutions, will drive us to effective techniques to solve all proposed challenges.

Keywords: mapping challenges, ShExML, data mapping languages, knowledge graph construction

1 Introduction

Mapping heterogeneous datasources using a single representation is an active field which has been getting traction in the past years. For this purpose a set of languages and tools has been proposed [8] which lower the cost and time employed in these tasks. This trajectory ended up in the celebration of the 1st International Workshop on Knowledge Graph Building [1] and, one year after, the beginning of the Knowledge Graph Construction W3C Community Group[1] where academia, industry and practitioners are gathered to envision new steps, find unsolved problems, and face new challenges in this field[2]. One of the community outputs was a set of mapping challenges[3] which are nowadays complicated to solve with the current state-of-the-art languages, tools and techniques.

[1] https://w3id.org/kg-construct/tpac/
[2] https://w3id.org/kg-construct/tpac/#report
[3] https://w3id.org/kg-construct/workshop/challenges.html

Therefore, in this paper we tackle some of these mapping challenges with ShExML [5] and try to solve the following questions:

- Q1: How can mapping challenges be solved with ShExML?
- Q2: How can unaddressed challenges be solved and implemented in ShExML?
- Q3: Have modifications in ShExML affected the functioning of already present features?

The rest of the paper is structured as follows: In Section 2 we summarise the mapping challenges proposed by the community; additionally, we offer a set of suplementary material to support following explanations. In Section 3 we see how the current language specification and engine can solve some of the challenges. Then, we explain how the solutions for other challenges have been implemented and included in ShExML in Section 4. In Section 5 we propose some further language modifications and a discussion on how the rest of the challenges could be addressed. We demonstrate the old features integrity after including the new ones and we establish a discussion on mapping challenges results in Section 6. And, finally, in Section 7 we draw some conclusions.

2 Mapping challenges summary

During consecutive meetings in the Knowledge Graph Construction W3C Community Group, several mapping challenges and problems were arisen which are collected in the workshop website[4]. Thus, in this paper we deal with this selection of mapping challenges, we examine them and propose solutions within the ShExML language and our engine[5]. To categorise these solutions we link them to ShExML versions so we can trace when these solutions were achieved, i.e., if they were possible to solve before the mapping challenges were defined (ShExML v0.2.3), if they were solved after the mapping challenges were defined (ShExML v0.2.4 & v0.2.5) or if they are not yet solved (future versions).

Thus, in Table 1 a summary table is offered with the addressed challenges and with which version of ShExML engine the expected output is achieved. Besides, we offer a webpage[6] with links to the working solutions as supplementary material for the sake of demonstration and reproducibility. The inputs are taken from the Knowledge Graph Construction W3C Community Group mapping challenges repository[7] which contains a set of inputs and expected outputs that are agreed to represent the community mapping problems.

In the following sections we explain how solutions are achieved, which ShExML constructions and techniques were used, we establish a discussion on reached solutions and how unsolved challenges could be addressed in ShExML.

[4] https://w3id.org/kg-construct/workshop/challenges.html
[5] https://github.com/herminiogg/ShExML
[6] http://herminiogg.github.io/mapping-challenges/challenges/solutions.html
[7] https://github.com/kg-construct/mapping-challenges

		Already solved (v0.2.3)	With language modifications (v0.2.4 & v0.2.5)
Access fields outside iterators	input 1	**x**	✓
	input 2	**x**	**x**
Datatype map	input 1	**x**	✓
	input 2	**x**	✓
	input 3	**x**	✓
	input 4	**x**	✓
	input 5	✓	✓
Excel style	input 1 (unaddressed)	**x**	**x**
Generate multiple values	input 1	**x**	✓
	input 2	**x**	✓
Join on literal	input 1	✓	✓
Language map	input 1	**x**	✓
	input 2	**x**	✓
	input 3	**x**	✓
Multivalue references	input 1	**x** (bug)	✓
Process multivalue references	input 1 (unaddressed)	**x**	**x**
RDF Collections	input 1	**x**	✓
	input 2	**x**	**x**
	input 3	**x**	**x**

Table 1. Coverage table of mapping challenges in ShExML language and engine by input files. ✓means covered and **x** means not covered. Unaddressed means that this challenge was not tried to solve in this work.

3 Already solved mapping challenges

In this section, we deal with the mapping challenges that can be solved without any modification in the ShExML language and engine. Therefore, these solutions are those that are reachable with ShExML v0.2.3 (released on 29th October 2020)[8], that is, before the mapping challenges were defined.

3.1 Datatype map (input 5)

Datatype map refers to the possibility to generate datatype tags from the input content. Therefore, instead of defining them statically in the mapping rules, this challenge aims to support the dynamic generation of datatype tags from input content. In the case of input 5, it is intended that mapping languages would be able to generate datatype tags according to the most probable value according to values formats. For example, in input 5 it is expected that the number 3 would have an `xsd:integer` datatype and 3.14 would have an `xsd:decimal` one.

This inference mechanism was already implemented in ShExML engine which in case that the user does not specifically define a datatype for an object value it will infer the most probable one (see Listing 1.1). Although the current implementation solves this specific mapping challenge, it is a naïve implementation.

[8] https://github.com/herminiogg/ShExML/releases/tag/v0.2.3

However, it can lead to a more complex inference system, for example, aligning existing input data sources datatypes with RDF ones.

Listing 1.1. ShExML datatype inference function.

```
protected def searchForXSDType(o: String): RDFDatatype = {
  if(Try(o.toInt).isSuccess)
    XSDDatatype.XSDinteger
  else if(Try(o.toDouble).isSuccess)
    XSDDatatype.XSDdecimal
  else if(Try(o.toBoolean).isSuccess)
    XSDDatatype.XSDboolean
  else
    XSDDatatype.XSDstring
}
```

3.2 Join on literal

This challenge refers to the possibility to generate literals from a join condition (i.e., from another source) where R2RML[9] and RML [3] output a resource by default.

In ShExML, join conditions[10] generate values without any specific form, so it is not determined in this step if it is a literal or a resource. It is, then, defined by the user in the shapes part, where the user decides the form of the output. This is a design decision on ShExML that was driven by the separation of concerns main principle. In Listing 1.2 we can see how the join condition is defined in `familyName` expression, and how then this expression is used in `:Author` shape without any prefix, indicating that a literal must be generated.

Listing 1.2. ShExML solution for join on literals.

```
PREFIX : <http://example.com/>
PREFIX experson: <http://example.com/person/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX schema: <http://schema.org/>
SOURCE jsonfile <https://raw.githubusercontent.com/
kg-construct/mapping-challenges/
2aac9680cd731fd647abd33d44a7f400e4278cf3/
challenges/join-on-literal/input-1/input.json>
ITERATOR author <jsonpath: $.author[*]> {
  FIELD id <id>
  FIELD firstname <firstname>
  FIELD affiliation <affiliation>
}
ITERATOR people <jsonpath: $.people[*]> {
```

[9] https://www.w3.org/TR/r2rml/

[10] See ShExML specification for a full explanation on how join construction works: http://shexml.herminiogarcia.com/spec/#join

```
  FIELD firstname <firstname>
  FIELD familyname <familyName>
}
EXPRESSION authors <jsonfile.author UNION jsonfile.people>
EXPRESSION familyName <jsonfile.people.familyname UNION
  jsonfile.author.firstname JOIN jsonfile.people.firstname>

:Author experson:[authors.id] {
  :affiliation [authors.affiliation] ;
  :lastName [familyName] ;
}
```

3.3 Multivalue references

This challenge deals with the expected output of a hierarchical document (e.g., XML or JSON files) where multiple iterators are used. The discussion in this challenge is whether we produce the cartesian product and provide a join condition to correlate values or if we just translate the hierarchical information as it is, without the need to provide any join condition[11]. This case becomes more complicated if a join condition needs to be provided over a JSON file because of the impossibility to access parent nodes (see Section 4.1 for the specific challenge on this topic). Therefore, it seems that in hierarchical data the expected output should be a verbatim translation (so to say, not creating the cartesian product as it is not how it is originally represented in the input file).

In ShExML, this was the default behaviour from its inception as in ShExML first versions it only supported XML and JSON files. Besides, we saw it as a more usable manner to define these mappings as usability is the main goal of the language [4]. Therefore, in Listing 1.3 we can see how using iterators and nested iterators we can cover these hierarchical data models. In Table 1 this challenge is marked as not solved in ShExML v0.2.3 due to a bug when using only the root node (`$`) in the top iterator query. However, we include it here as the coverage of this challenge did not require syntax modifications nor new features in ShExML engine, only a bug fix.

Listing 1.3. ShExML solution multivalue references.

```
PREFIX ex: <http://example.com/>
PREFIX exLab: <http://example.com/lab/>
PREFIX exArticle: <http://example.com/article/>
PREFIX exAuthor: <http://example.com/author/>
PREFIX exAff: <http://example.com/aff/>
SOURCE lab_file <https://raw.githubusercontent.com/
kg-construct/mapping-challenges/main
/challenges/multivalue-references
/input-1/input.json>
```

[11] See discussion about this topic in RMLMapper reference implementation: https://github.com/RMLio/rmlmapper-java/issues/28

```
ITERATOR lab <jsonpath: $> {
  FIELD labName <labName>
  ITERATOR articles <articles[*]> {
    FIELD title <title>
    ITERATOR authors <authors[*]> {
      FIELD name <name>
      ITERATOR affiliation <affiliation[*]> {
        FIELD label <label>
      }
    }
  }
}

EXPRESSION labValues <lab_file.lab>

ex:Lab exLab:[labValues.labName] {
  a ex:Lab ;
  ex:hasArticles @ex:Article ;
  ex:hasMembers @ex:Author ;
}

ex:Article exArticle:[labValues.articles.title] {
  ex:hasAuthor @ex:Author ;
}

ex:Author exAuthor:[labValues.articles.authors.name] {
  ex:hasAffiliation
   exAff:[labValues.articles.authors.affiliation.label] ;
}
```

4 Language modifications

In this section, we deal with the mapping challenges that needed some modifications in the ShExML language and engine. Therefore, these solutions are those which are reachable with ShExML v0.2.4 (released on 18th January 2021)[12] or with ShExML v0.2.5 (released on 27th January 2021)[13], that is, after the mapping challenges were defined.

4.1 Access fields outside iterators

Sometimes, in hierarchical data models, there is the need to access values outside the iteration pattern. For example, we may need to obtain the values that are parents of the current iterated node. When dealing with XML files it does not involve any modification in ShExML, as using XPath queries we are able to

[12] https://github.com/herminiogg/ShExML/releases/tag/v0.2.4
[13] https://github.com/herminiogg/ShExML/releases/tag/v0.2.5

access upper nodes with the double dot and slash notation (i.e., `../`). However, when dealing with JSON files, this is not possible because of JSONPath not supporting the parent access notation[14].

This is a well-known problem in data mapping languages as they use JsonPath to define values accesses. Indeed, in xR2RML [11], the authors defined a property called `xrr:pushDown` that takes a value in the hierarchy and pushes it down into their offsprings iterators so it can be available further [12].

Following this experience with xR2RML, we implemented a similar solution in ShExML using the `PUSHED_FIELD` and `POPPED_FIELD` keywords. When using the `PUSHED_FIELD` keyword the ShExML engine saves the value using the name as the identifier for further uses. Then, when the `POPPED_FIELD` is used the ShExML engine searches for the saved value with an identifier which is equal to that given in the query part (i.e., inside `<` and `>`). Therefore, in Listing 1.4, the `id` field is saved and then used in the `cars` iterator, so we can establish a relation from the car to the owner.

An interesting discussion here is if it is better to make these accesses implicit or explicit. A recent RML syntax modification proposal [2] presented an algorithm that could implicitly access values from upper hierarchical levels by saving iteration information, indexes and values. Users can directly access upper elements from its current hierarchical level. Indeed, the solution is clean and avoids the user's explicit declaration of values to be saved. However, it has two possible main drawbacks. Firstly, the use of a bigger amount of memory and time by saving a lot information that could be used (or not) in further mapping rules. This technique, could be, in the end, a bottleneck in performance if it is not carefully implemented. Second one, it could complicate the engine implementation as it allows to go up and down in the hierarchy, while actual behaviour only expects to go down. This could also lead to a performance issue. Either way, this dichotomy should be quantified in further experiments to establish the best solution in terms of usability and performance.

Listing 1.4. ShExML solution for accessing fields outside iterations.

```
ITERATOR records <jsonpath: $.records[*]> {
  PUSHED_FIELD id <id>
  FIELD enteredBy <enteredBy>
  ITERATOR cars <cars[*]> {
    FIELD make <make>
    POPPED_FIELD carOwner <id>
  }
}
```

4.2 Datatype map

As we mentioned in Section 3.1, this challenge aims to generate datatype tags dynamically from data content. Therefore, the datatype inputs can appear in

[14] https://goessner.net/articles/JsonPath/

multiple ways: full URI, prefix plus datatype, or simply datatype name without prefix.

ShExML v0.2.3 supports the creation of static datatype tags with prefix plus datatype syntax (see Listing 1.5). Therefore, we should derive this syntax and maintain its proven usability [4] but giving dynamic datatype generation possibilities. The natural expansion of this syntax is to include the same object generation expression but also for datatypes and language tags (see Section 4.3). So, the final syntax is prefix plus generation expression (inside square brackets) as we can see in Listing 1.6. The prefix can be optional if the data value already contains it (e.g., input 1 and 2) and values can be transformed using Matcher feature[15] to expected XML Schema valid datatypes (e.g., input 4).

Listing 1.5. ShExML static datatypes syntax.

```
ex:Person exPerson:[person.firstname] {
  ex:num [person.num] xsd:integer ;
}
```

Listing 1.6. ShExML dynamic datatypes syntax.

```
ex:Person exPerson:[person.firstname] {
  ex:num [person.num] xsd:[person.dt] ;
}
```

4.3 Language map

As with datatype maps in Section 4.2, the language map challenge want to address the problem of generating language tags dynamically from input data. In ShExML v0.2.3 language tags were supported statically, that is, it was possible to tag an object expression with a specific language but it would be applied to all values (see Listing 1.7).

We performed a syntax and engine modification, like in datatype maps, to be able to generate language tags with expressions. The final syntax is `@` plus the generation expression (between square brackets) as it can be seen in Listing 1.8. Here, again, the idea was to preserve the usability as the main goal and to make it as simple as possible. Input 1 tests the generation with a valid tag following BCP47[16], input 2 tests the transformation of a language value to a valid tag (in ShExML this is done using Matchers functionality[17]), and in input 3 it is shown how two different sources can be joined to provide language information.

As a side note, it is not possible to specify a language map and a datatype map in the same triple, as it is forbidden by the ShExML grammar. This was made intentional to follow the RDF specification rules as it can be seen in its

[15] http://shexml.herminiogarcia.com/spec/#matcher
[16] https://tools.ietf.org/html/bcp47
[17] http://shexml.herminiogarcia.com/spec/#matcher

grammar[18]. Therefore, whenever a langtag generation expression (either static or dynamic) is provided the implicit datatype is `rdf:langString`.

Listing 1.7. ShExML static generation of language tags.

```
ex:Person exPerson:[person.firstname] {
  ex:lastName [person.lastname] @en ;
}
```

Listing 1.8. ShExML dynamic generation of language tags

```
ex:Person exPerson:[person.firstname] {
  ex:lastName [person.lastname] @[person.lang] ;
}
```

4.4 Generate multiple values

This challenge wants to address the problem of generating various datatypes or language tags for the same subject (e.g., a multi-language value). Once datatype maps (see Section 4.2) and language maps (see Section 4.3) are supported in ShExML, it is straightforward as ShExML will generate a triple per value returned from the object expression. Therefore, to generate multi-language values the syntax is like in Listing 1.9 and to generate multi-language values with a default language the syntax is like in Listing 1.10.

Listing 1.9. ShExML multiple values with language tags.

```
ex:Person exPerson:[person.lastname] {
  ex:name [person.firstname.label] @[person.firstname.lang] ;
}
```

Listing 1.10. ShExML multiple values with language tags and with a default language.

```
ex:Person exPerson:[person.firstname] {
  ex:name [person.firstname] @en ;
  ex:name [person.firstname] @[person.lang] ;
}
```

4.5 RDF Collections

This challenge puts on the table the necessity for a mechanism to create RDF Collections from some values. Normally, in ShExML, and in other data mapping languages, when an object generation expression returns multiple values, multiple triples are generated (see Section 3.3). However, in certain cases it is necessary to encapsulate these values inside a collection (e.g., to preserve order).

[18] https://www.w3.org/TR/turtle/#h3_sec-grammar-grammar

This was already explored by some languages (e.g., SPARQL-Generate [7] and xR2RML [11]) which provide some directives to create collections. Therefore, we applied this experience in ShExML to cover RDF Collections and Containers (i.e., Lists, Seqs, Bags and Alts.). Now, it is possible to indicate to the engine that a collection or container should be generated using the keyword `AS` plus the desired collection or container (i.e., `RDFList`, `RDFBag`, `RDFSeq` or `RDFAlt`). The proposed syntax follows the same design principles from already existing features syntax (e.g., Matchers[19]). See Listing 1.11 for an example.

Listing 1.11. ShExML support for RDF collections and containers.

```
ex:Article exArticle:[labValues.articles.title] {
  a ex:Article ;
  ex:hasAuthors
    exAuthor:[labValues.articles.authors.name AS RDFList] ;
}
```

5 Future required actions

In this section we discuss further challenges that are not solved with the previously mentioned modifications. These are the challenges that would require to rethink some functionality, or to include new ones, but that would need from a well planned inclusion, due to their possible interference with other features.

5.1 Access fields outside iterators (input 2)

Although this challenge was already addressed in Section 4.1, only input 1 was completely solved. In the case of input 2, where data is in the same hierarchical level (like it would come from two different files), using join conditions in ShExML, only one car is linked to each owner when the expected result was two cars per person. To solve this problem we think of two possible solutions.

First one is to review the join condition functionality to check whether something is failing (a bug) or if the join condition need to be rethought and reimplemented to cover further challenges.

Another possibility, which is already present in other languages like YARRRML [6], is to provide conditional generation. With conditional content generation we are able to test a condition (e.g., in input 2 for value equality) and generate or not the resulting triple depending on its result.

5.2 Excel style

A classic solution when dealing with Excel sheets was to convert them to CSV, and then treat them as tables to be processed by data mapping languages. However, this challenge found this solution not appropriate when the style of the Excel sheet is wanted to be preserved. Two solutions could cover this challenge.

[19] http://shexml.herminiogarcia.com/spec/#matcher-0

First one is to preprocess the Excel sheet and convert it to CSV but adding columns with style information so it can be processed by state-of-the-art tools. However, it would require some preprocessing work which would weaken the goal of low cost and time invested when using data mapping tools.

Second one is to include a specific Excel processor, with its own query language, which can express not only access to cells, but also to the cell and text style. Thus, in Java based implementations it can be considered to use Apache POI to process sheets, and include some simple query support to retrieve styles. Therefore, in ShExML this would require to support the mentioned query language for Excel, and then, integrate it into the ShExML engine to retrieve Excel sheets values.

5.3 Process multivalue references

This challenge is very close to multivalue references (see Section 3.3), but in this case multivalues are included all within a string value and separated by commas.

Therefore, here the challenge is not about how to output multivalues, or create RDF collections, but how to effectively handle these multivalues which need some processing. Therefore, this would require some sort of data transformations functions that can be applied to the extracted values. Therefore, the most effective way to extend ShExML and enable users to transform data is to provide the possibility to execute transformation functions which can be defined by users.

Data transformation functions have been already explored in RML through the FnO library [9] which provides a set of implementation independent reusable functions [10]. So, one possibility is to support FnO functions inside ShExML. The advantage of this proposal is that it moves all function infrastructure outside the ShExML language and engine. Conversely, we add more dependencies to users (which can find it hard to learn), we force them to use a third party environment and we lose control of this part.

Another possibility is to provide an environment to define inline functions like the semantic actions in Shape Expressions (ShEx) [13]. Therefore, we can provide a restricted environment where higher order functions could be executed (see Listing 1.12 for an example). The advantages are that there is no need for third party dependencies, it provides a higher flexibility and users do not need to learn another tool. However, it can increase complexity due to the necessity to know about functional programming.

Listing 1.12. ShExML support for RDF collections and containers.

```
PREFIX ex: <http://example.com/>
SOURCE lab_file <https://raw.githubusercontent.com/
kg-construct/mapping-challenges/main/challenges/
process-multivalue-references/input-1/input.json>
FUNCTION splitFunction <n => n.split(',')>
ITERATOR lab <jsonpath: $> {
  FIELD labName <labName>
  ITERATOR articles <article> {
```

```
    FIELD title <title>
    FIELD tags <tags>
  }
}

EXPRESSION labValues <lab_file.lab>

ex:Tag ex:[lab.articles.tag WITH splitFunction] {
  ex:label [lab.articles.tag WITH splitFunction] ;
}
```

5.4 RDF Collections (input 2 & 3)

Although RDF collections and containers were included in ShExML (see Section 4.5), input 2 and 3 present some particularities. In the case of input 2 the use of different keys would require a more complex query or some sort of parametrisation in executed queries. In input 3 the per row iteration model for CSV files implemented in ShExML does not create collections effectively. Therefore, it would imply a reimplementation of per row iteration model for these cases. However, it could affect the overall functionality for CSV files.

6 Evaluation and Discussion

In Q3 we have posed a question about the possible effects that the modifications in ShExML could have in already working features. The idea of this question is to demonstrate the validity of Q1 solutions alongside old features that should still work as expected. This type of testing, known as regression testing, have been included in ShExML from the very beginning[20] so we are able to add new features in ShExML knowing that old features are still working as expected. Thus, every time a new version is released these tests must be executed to validate the language and engine integrity. Continuous integration is the perfect tool for this task, as every time that a change is submitted to ShExML repository all tests are executed to verify the integrity. In the ShExML repository we have configured Travis CI[21] for this task. Therefore, these regression tests in v0.2.4[22] and v0.2.5[23] are telling us that all features are still working as expected, and equally, giving a negative answer to Q3. So, we can conclude that integrity is held.

In Sections 3 and 4 we have seen how some mapping challenges were already solved in ShExML and how we have made some modifications in ShExML language and engine to deal with the others. These two sections give an answer

[20] To see all tests that are executed over ShExML engine https://github.com/herminiogg/ShExML/tree/master/src/test/scala-2.12/es/weso/shexml

[21] https://travis-ci.org/

[22] https://travis-ci.org/github/herminiogg/ShExML/builds/755033209

[23] https://travis-ci.org/github/herminiogg/ShExML/builds/756419674

for Q1. These solutions were designed to maintain ShExML usability [4] using a similar syntax, so that users can use these new features with the minimum learning curve possible; in other words, making the smallest modifications in the ShExML syntax. In addition, in Section 3 we have highlighted how the ShExML design have already given an answer to some challenges, emphasising how the ShExML separation of concerns principle can give answers to some of them (e.g., Join on literal).

In Section 5 we have given some intuition on how remaining challenges could be solved, answering to Q2. They would require harder and more complex modifications; in some cases the modification of an already working mechanism (e.g., inputs 2 and 3 in RDF Collections), the inclusion of a new iteration model and the design of a new query language (e.g., Excel style) or the choice between two different systems (e.g., data transformation functions in process multivalue references). All these inclusions will require a careful study and implementation in the language so they do not affect other features and, also, to select the better option from a usability perspective.

7 Conclusions

In this paper we have explored how ShExML can deal with some of the challenges defined in the Knowledge Graph Construction W3C Community Group. We have divided them into challenges already solved by ShExML before their definition, challenges solved by latest versions of ShExML, and challenges that are not yet solved, for which we have given some notions and intuitions on how ShExML can be modified to cover them. Furthermore, we have demonstrated that the modification of ShExML to cover new challenges has not affected other language and engine features. Therefore, we see this work as a first step on demonstrating how the challenges can be solved and, together with solutions from other languages and the joint discussion, we will be able to offer unified solutions to the posed mapping challenges.

References

1. Chaves-Fraga, D., Heyvaert, P., Priyatna, F., Sequeda, J.F., Dimou, A., Jabeen, H., Graux, D., Sejdiu, G., Saleem, M., Lehmann, J. (eds.): Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia, June 3, 2019, CEUR Workshop Proceedings, vol. 2489. CEUR-WS.org (2019)
2. Delva, T., Van Assche, D., Heyvaert, P., De Meester, B., Dimou, A.: Integrating nested data into knowledge graphs with RML fields. In: To appear on: Proceedings of the 2nd International Workshop on Knowledge Graph Building co-located with 18th Extended Semantic Web Conference (ESWC 2021), Hersonissos, Greece, June 6, 2021. CEUR Workshop Proceedings (2021)
3. Dimou, A., Sande, M.V., Colpaert, P., Verborgh, R., Mannens, E., de Walle, R.V.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data.

In: Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014. (2014)
4. García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C.: ShExML: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Computer Science **6**, e318 (2020)
5. García-González, H., Fernández-Álvarez, D., Gayo, J.E.L.: ShExML: An Heterogeneous Data Mapping Language based on ShEx. In: Proceedings of the EKAW 2018 Posters and Demonstrations Session co-located with 21st International Conference on Knowledge Engineering and Knowledge Management (EKAW 2018), Nancy, France, November 12-16, 2018. pp. 9–12 (2018)
6. Heyvaert, P., Meester, B.D., Dimou, A., Verborgh, R.: Declarative Rules for Linked Data Generation at Your Fingertips! In: The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers. pp. 213–217 (2018)
7. Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL Extension for Generating RDF from Heterogeneous Formats. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10249, pp. 35–50 (2017)
8. Meester, B.D., Heyvaert, P., Verborgh, R., Dimou, A.: Mapping Languages: Analysis of Comparative Characteristics. In: Chaves-Fraga, D., Heyvaert, P., Priyatna, F., Sequeda, J.F., Dimou, A., Jabeen, H., Graux, D., Sejdiu, G., Saleem, M., Lehmann, J. (eds.) Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia, June 3, 2019. CEUR Workshop Proceedings, vol. 2489, pp. 37–45. CEUR-WS.org (2019)
9. Meester, B.D., Maroy, W., Dimou, A., Verborgh, R., Mannens, E.: RML and FnO: Shaping DBpedia Declaratively. In: Blomqvist, E., Hose, K., Paulheim, H., Lawrynowicz, A., Ciravegna, F., Hartig, O. (eds.) The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10577, pp. 172–177. Springer (2017)
10. Meester, B.D., Seymoens, T., Dimou, A., Verborgh, R.: Implementation-independent function reuse. Future Gener. Comput. Syst. **110**, 946–959 (2020)
11. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of Relational and Non-relational Databases into RDF with xR2RML. In: Monfort, V., Krempels, K., Majchrzak, T.A., Turk, Z. (eds.) WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015. pp. 443–454. SciTePress (2015)
12. Michel, F., Djimenou, L., Zucker, C.F., Montagnat, J.: xR2RML: Relational and non-relational databases to RDF mapping language. Tech. rep. (2017)
13. Prud'hommeaux, E., Gayo, J.E.L., Solbrig, H.R.: Shape expressions: an RDF validation and transformation language. In: Sack, H., Filipowska, A., Lehmann, J., Hellmann, S. (eds.) Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014. pp. 32–40. ACM (2014)