# Why to Tie to a Single Data Mapping Language? Enabling a Transformation from ShExML to RML

Herminio García-González[1,*], Anastasia Dimou[2]

[1]Kazerne Dossin: Memorial, Museum and Research Centre on Holocaust and Human Rights, Mechelen, Belgium
[2]KU Leuven, Department of Computer Science, Sint-Katelijne-Waver, Belgium

**Abstract**

Different mapping languages (e.g., RML, SPARQL-Generate and ShExML) have appeared in the last years covering different use cases, scenarios and functionalities. However, users cannot seamlessly interchange between these mapping languages. In this paper, we propose a translation from ShExML to RML letting users benefit from the usability of ShExML and the wide-support and functionalities of RML.

**Keywords**

declarative mapping rules, translation, data mapping languages, ShExML, RML

## 1. Introduction

Declarative mapping languages have been increasingly adopted during the last years in different fields (e.g., DBpedia [1], digital heritage [2] or the railway domain [3])[1]. These mapping languages can be used to specify declarative mapping rules which allow for a modifiable, flexible, repeatable and shareable workflow when integrating heterogeneous data sources in a Knowledge Graph (KG) [4] superseding the imperative solutions. However, not all mapping languages cover the same functionalities, leading to diverse mapping languages. The consequent lack of interoperability between the different languages ties users to a certain specification, preventing them to switch between mapping languages.

Among the different declarative mapping languages, RML [5] is the one that has been the most adopted by the community. More than 20 associated systems implement RML allowing different approaches and optimisations[2] which make RML a very reliable and long-term solution. However, its syntax, based on RDF, tends to be very verbose and thus not so friendly for humans. In this context, ShExML appeared with a clear vocation on usability letting users be more productive when developing mapping rules [6]. However, its specification only counts with a conformant engine[3], which does not cover all the functionalities that users could face nor can users take advantage of all the optimisations offered by the different RML implementations.

*SEMANTICS 2022 EU: 18th International Conference on Semantic Systems, September 13-15, 2022, Vienna, Austria*
*Corresponding author.
herminio.garciagonzalez@kazernedossin.eu (H. García-González); anastasia.dimou@kuleuven.be (A. Dimou)
https://herminiogarcia.com/ (H. García-González); https://natadimou.com (A. Dimou)
0000-0001-5590-4857 (H. García-González); 0000-0003-2138-7972 (A. Dimou)


[1]See more on: https://github.com/kg-construct/use-cases
[2]https://github.com/kg-construct/resources/blob/master/tools.md
[3]https://github.com/herminiogg/ShExML

Thus, it is important to build bridges to foster adoption and trust in the declarative KG construction ecosystem while supporting users to effectively and seamlessly cover their needs. With this objective in mind, in this paper, we propose a conversion from ShExML to RML, letting users rapidly sketch their mapping rules in ShExML and then switch to RML to take advantage of its plethora of implementations. A live demo is available at the ShExML playground[4].

## 2. Related Work

Different mapping languages were proposed in the past [7] based on *dedicated languages*, e.g., RML extending R2RML to support heterogeneous data sources; *repurposed languages*, e.g., SPARQL-Generate [8] extending the SPARQL query language or ShExML using Shape Expressions (ShEx)-based syntax [9]; or *serialisation languages* like YARRRML [10] extending YAML. As YARRRML is a tightly coupled serialisation for RML aiming for better readability by humans, translating YARRRML to RML is straightforward[5]. Besides the YARRRML to RML translation, an RML to SPARQL-Generate[6] translation exists enabling the use of RML mapping rules with the SPARQL-Generate implementation, but the inverse translation is not tackled.

Recently, an ontology was presented as a meta-language to represent the functionalities of existing mapping languages [11]. Future implementations will offer translations from and to the targeted mapping languages. Despite covering all mapping rules, being a meta-language involves more verbose constructions than existing languages, surpassing even a verbose language like RML. In addition, covering the expressiveness from all potential mapping languages could be a difficult—if ever approachable—task. Thus, specific translations are still necessary.

In this paper we propose a specific translation from ShExML to RML to combine the usability of ShExML and the sustainability and wide-support of RML implementations.

## 3. Brief introduction of RML and ShExML

The two languages offer different syntaxes for integrating heterogeneous data into a KG. The example in Figure 1 generates a triple for each entity, showing their correspondences.

On the one hand, RML[7] offers a syntax based in RDF which ties the representation of the language to the RDF serialisation formats (i.e., Turtle, NTriples, RDF/XML, etc.). For describing the mapping process it relies on classes like: `rr:SubjectMap`, `rr:PredicateMap` and `rr:ObjectMap` that define how the subject, predicate and object of a triple will be generated; `rr:PredicateObjectMap` which relates a predicate with an object; `rml:LogicalSource` which describes an input data source; and `rr:TriplesMap` which defines how a set of triples will be generated for a certain data source. Each class offers different fields to modify the aspects of the generation.

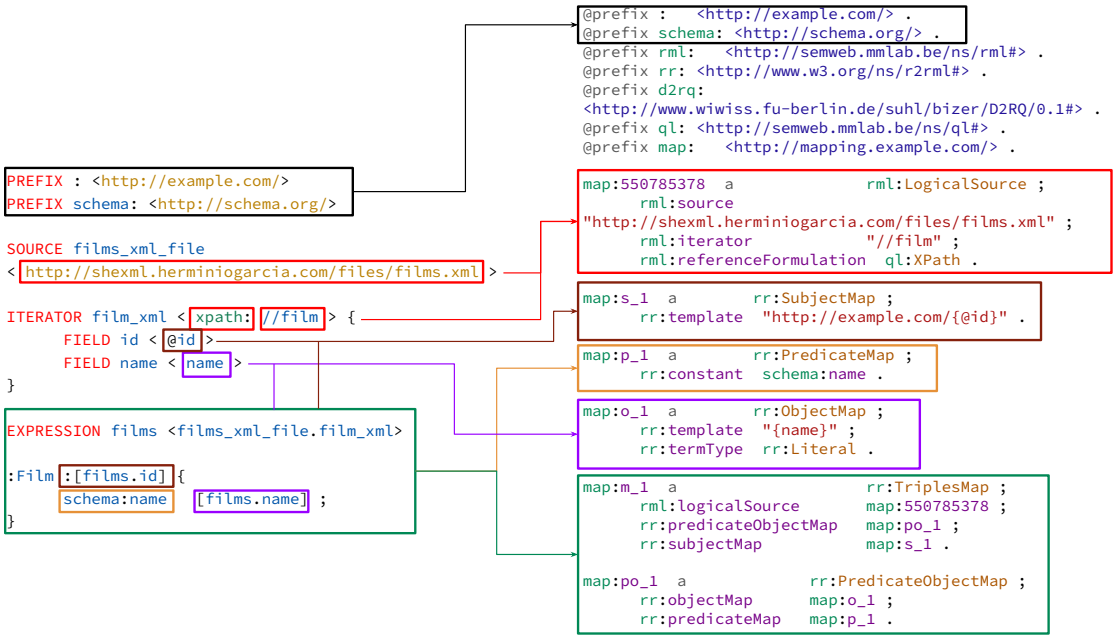
On the other hand, ShExML[8] separates the constructions intended to extract values (declarations) from those to generate triples (shapes). ShExML offers: `SOURCE` which points to a data

[4]http://shexml.herminiogarcia.com/editor/
[5]https://github.com/RMLio/yarrrml-parser
[6]https://github.com/sparql-generate/rml-to-sparql-generate
[7]https://rml.io/specs/rml/
[8]http://shexml.herminiogarcia.com/spec/

source, ITERATOR which defines a query whose results need to be iterated, FIELD that holds the query to a value that will be outputted in a triple, and EXPRESSION which relates the sources with the iterator and field queries. Then, the shapes part allows to format the output in triples using already defined expressions for subjects, predicates and objects.

## 4. Translating ShExML constructions to RML rules

**Figure 1:** A set of ShExML mapping rules (left side) and an equivalent set of RML mapping rules (right side). Each color identifies a correspondence between an RML construction and a ShExML construction.




As ShExML follows a ShEx-based syntax, a dedicated parser fitting the ShExML grammar is required. For this purpose, the translator is embedded in the ShExML engine, taking advantage of the already developed modules. Once the abstract tree is generated the translator traverses it to generate the triples that conform an RML rules set. This process is similar to how the ShExML engine outputs the mapped RDF but this time it generates RML. The translator performs a one to one translation of the different directives, taking into account that in many cases information in ShExML rules is dispersed across different directives in comparison with their RML counterparts (see Figure 1 for a diagram on how ShExML constructions are translated to RML). A live demo of this algorithm can be tested on the ShExML playground[9]. To verify the validity of the performed translations we developed a set of test cases[10] that cover different aspects of ShExML functionality. The test cases convert the ShExML rules to RML rules. The latter are executed in the rml-mapper processor[11] and the output is compared to the output of the ShExML engine.

[9]http://shexml.herminiogarcia.com/editor/
[10]https://github.com/herminiogg/ShExML/tree/master/src/test/scala-2.12/com/herminiogarcia/shexml/rml
[11]https://github.com/RMLio/rmlmapper-java

### Non translatable directives

Even though many declarative mapping languages share the same foundations and a common iteration model [11], they have evolved in different directions covering different functionalities. Therefore, unlike YARRRML, not all ShExML features can be directly translated to RML statements, or it would involve too complex processes to make them possible.

Certain features are included in ShExML but they are not directly supported in RML. This includes Matchers[12] (substitute one string occurrence for other string), String operations[13] (concatenate two strings), and Auto-increment ids[14]. All these functionalities could be covered in RML using the FnO [12] extension but that is beyond the scope of this translation. Last year, ShExML introduced a set of modifications to cope with the mapping challenges[15] [13]. It is now possible to define dynamic data types[16] and language tags[17]. This has been tackled by RML specification for dynamic language tags generation but not for dynamic datatypes. ShExML allows accessing fields outside the iteration scope. This is useful when accessing parent nodes in JSON as JSONPath specification does not support it[18]. It has been described how to tackle this with RML [14] but it is not yet implemented in the specification.

Finally, external functions execution has been recently supported in ShExML. However, ShExML uses Scala classes that can be directly executed while FnO relies on a function hub where functions can be implemented in JavaScript or Java. This could be resolved by translating from Scala to Java code and then pushing the Java code to the Function Hub [12].

## 5. Conclusions

In this work, we described a translation from ShExML to RML, letting users sketch mapping rules rapidly in ShExML and then switching to RML. In addition, we discussed the limitations of the current translation due to different coverage of features between the two languages. In the future, we will tackle the inverse translation, i.e., from RML to ShExML. This will let more users adopt ShExML using their already created RML rules as well as allowing them to switch back and forth between the two languages depending on their needs.

## Acknowledgments

This work was carried out in the context of the EHRI-3 project funded by the European Commission under the call H2020-INFRAIA-2018-2020, with grant agreement ID 871111 and DOI 10.3030/871111.

[12] http://shexml.herminiogarcia.com/spec/#matcher
[13] http://shexml.herminiogarcia.com/spec/#string-operation
[14] http://shexml.herminiogarcia.com/spec/#autoincrement-ids
[15] https://kg-construct.github.io/workshop/2021/challenges.html
[16] http://shexml.herminiogarcia.com/spec/#data-types-dynamic-version
[17] http://shexml.herminiogarcia.com/spec/#lang-tags-dynamic-version
[18] https://goessner.net/articles/JsonPath/

## References


[1] B. De Meester, W. Maroy, A. Dimou, R. Verborgh, E. Mannens, Declarative data transformations for Linked Data generation: the case of DBpedia, in: The Semantic Web – ISWC 2017, Springer, 2017. doi:10.1007/978-3-319-68204-4_28.

[2] H. García-González, E. Albarrán-Fernández, J. E. L. Gayo, M. Calleja-Puerta, Converting Asturian Notaries Public deeds to Linked Data Using TEI and ShExML, in: Proceedings of the Third Workshop on Humanities in the Semantic Web, CEUR, 2020.

[3] J. A. Rojas, M. Aguado, P. Vasilopoulou, I. Velitchkov, D. V. Assche, P. Colpaert, R. Verborgh, Leveraging Semantic Technologies for Digital Interoperability in the European Railway Domain, in: The Semantic Web–ISWC, Springer, 2021. doi:10.1007/978-3-030-88361-4_38.

[4] B. D. Meester, P. Heyvaert, R. Verborgh, A. Dimou, Mapping Languages: Analysis of Comparative Characteristics, in: Joint Proceedings of the 1st Workshop on Knowledge Graph Building and 1st Workshop on Large Scale RDF Analytics, CEUR, 2019.

[5] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, R. V. de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: Proceedings of the Workshop on Linked Data on the Web, CEUR, 2014.

[6] H. García-González, I. Boneva, S. Staworko, J. E. Labra-Gayo, J. M. C. Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, PeerJ Computer Science (2020) e318. doi:10.7717/peerj-cs.318.

[7] D. V. Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. D. Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: a Systematic Literature Review, Journal of Web Semantics (2022). (In press).

[8] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: The Semantic Web–ESWC, Springer, 2017. doi:10.1007/978-3-319-58068-5_3.

[9] E. Prud'hommeaux, J. E. Labra Gayo, H. Solbrig, Shape Expressions: An RDF Validation and Transformation Language, in: Proceedings of the 10th International Conference on Semantic Systems, ACM, 2014. doi:10.1145/2660517.2660523.

[10] P. Heyvaert, B. De Meester, A. Dimou, R. Verborgh, Declarative rules for linked data generation at your fingertips!, in: The Semantic Web: ESWC Satellite Events, Springer, 2018. doi:10.1007/978-3-319-98192-5.

[11] A. Iglesias-Molina, A. Cimmino, E. Ruckhaus1, D. Chaves-Fraga, R. García-Castro, O. Corcho, An Ontological Approach for Integrating Declarative Mapping Languages, Semantic Web (2022). (In press).

[12] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent function reuse, Future Generation Computer Systems (2020). doi:10.1016/j.future.2019.10.006.

[13] H. García-González, A ShExML Perspective on Mapping Challenges: Already Solved Ones, Language Modifications and Future Required Actions, in: Proceedings of the 2nd International Workshop on Knowledge Graph Construction, CEUR, 2021.

[14] T. Delva, D. V. Assche, P. Heyvaert, B. D. Meester, A. Dimou, Integrating Nested Data into Knowledge Graphs with RML Fields, in: Proceedings of 2nd Workshop on Knowledge Graph Construction, CEUR, 2021.