RML2SHACL: RDF Generation Is Shaping Up

Thomas Delva
Thomas.Delva@ugent.be
Ghent University – imec – IDLab
Ghent, Belgium

Birte De Smedt
KU Leuven,
Department of Computer Science
Leuven, Belgium

Sitt Min Oo
X.SittMinOo@ugent.be
Ghent University – imec – IDLab
Ghent, Belgium

Dylan Van Assche
Dylan.VanAssche@ugent.be
Ghent University – imec – IDLab
Ghent, Belgium

Sven Lieber
Sven.Lieber@ugent.be
Ghent University – imec – IDLab
Ghent, Belgium
Sven.Lieber@kbr.be
KBR Royal Library of Belgium
Brussels, Belgium

Anastasia Dimou
Anastasia.Dimou@kuleuven.be
KU Leuven,
Department of Computer Science
Leuven, Belgium

ABSTRACT

RDF graphs are often generated by mapping data in other (semi-) structured data formats to RDF. Such mapped graphs have a repetitive structure defined by (i) the mapping rules and (ii) the schema of the input sources. However, this information is not exploited beyond its original scope. SHACL was recently introduced to model constraints that RDF graphs should validate. SHACL shapes and their constraints are either manually defined or derived from ontologies or RDF graphs. We investigate a method to derive the shapes and their constraints from mapping rules, allowing the generation of the RDF graph and the corresponding shapes in one step. In this paper, we present RML2SHACL: an approach to generate SHACL shapes that validate RDF graphs defined by RML mapping rules. RML2SHACL relies on our proposed set of correspondences between RML and SHACL constructs. RML2SHACL covers a large variety of RML constructs, as proven by generating shapes for the RML test cases. A comparative analysis shows that shapes generated by RML2SHACL are similar to shapes generated by ontology-based tools, with a larger focus on data value-based constraints instead of schema-based constraints. We also found that RML2SHACL has a faster execution time than data-graph based approaches for data sizes of 90MB and higher.

CCS CONCEPTS

• **Information systems** → *Data exchange*; *Extraction, transformation and loading*; *Graph-based database models*; **Resource Description Framework (RDF)**.

KEYWORDS

RML, R2RML, SHACL, RDF shapes, Shape generation

ACM Reference Format:
Thomas Delva, Birte De Smedt, Sitt Min Oo, Dylan Van Assche, Sven Lieber, and Anastasia Dimou. 2021. RML2SHACL: RDF Generation Is Shaping Up. In *Proceedings of the 11th Knowledge Capture Conference (K-CAP '21), December 2–3, 2021, Virtual Event, USA.* ACM, OnlineA, 8 pages. https://doi.org/10.1145/3460210.3493562

1 INTRODUCTION

RDF graphs are often generated by mapping data in other structures and formats to an RDF representation. There are different approaches to mapping non-RDF data to RDF, among which direct mapping [1] and customized mapping, e.g., [7, 8]. In *direct mapping*, an RDF representation is automatically generated from a non-RDF source; RDF terms in this representation are derived from the source's structure and content according to the direct mapping specification. In *customized mapping*, declarative mapping rules are used to define a custom RDF representation for one or more data sources. Such rules specify which RDF vocabulary terms are applied to the non-RDF sources. The structure of an RDF graph generated with customized mapping can be derived from the mapping rules: its structure is determined by (i) the structure of the non-RDF sources they are based on, and (ii) the mapping rules that define the sources' RDF representation.

The shape of an RDF graph, which is defined nowadays using e.g., the W3C recommended shape language, SHACL [12], is often desired. Shapes can be (i) *manually defined* using e.g., editors [3, 15], being time-consuming and error prone, (ii) *mined from the RDF graph* [2, 9, 21, 23], requiring the processing of large knowledge graphs, or (iii) *derived from the graph's ontology* [5, 6, 19], being limited to constraints that can be encoded in the ontology.

We propose an approach to derive the shapes of RDF graphs which are generated with mapping rules and make that shapes explicit using SHACL. The data owners can then define the mapping rules to generate the RDF graph and derive the shapes in one step. Our approach can derive (i) the logical schema of a mapped RDF graph (as when deriving shapes from the ontology), as well as (ii) data-level restrictions such as URI patterns (as when deriving the shape from the RDF graph).

In this paper, we present RML2SHACL: an approach to generate SHACL shapes that validate an RDF graph that was generated

K-CAP '21, December 2–3, 2021, Virtual Event, USA.

ACM ISBN 978-1-4503-8457-5/21/12…$15.00
https://doi.org/10.1145/3460210.3493562

from RML mapping rules. RML [8] is a generalisation of the W3C-recommended mapping language R2RML [7] that makes the mapping language extensible towards integrating other formats than relational databases. Our approach is applicable both to RML and R2RML, since we only rely on the parts shared between those two languages; we will refer to the intersection of R2RML and RML with the name "[R2]RML". RML2SHACL relies on a set of correspondences between [R2]RML and SHACL constructs we propose. We provide a Python implementation of RML2SHACL and report on initial experiments that confirm this implementation's feasibility.

Our contributions in this paper are: (i) a set of correspondences between [R2]RML and SHACL terms; (ii) an approach to generate SHACL from [R2]RML mapping rules that integrate data; (iii) an algorithm and a proof of concept implementation of our approach; (iv) a validation of correctness and proof of complete support for RML based on the RML test cases; and (v) comparative evaluations with ontology-based and RDF-graph-based tools.

The remainder of this paper is structured as follows. In section 2 we discuss related work on shape generation and preliminaries on RML and SHACL. In section 3 we propose our approach for generating shapes from mapping rules. In section 4 we discuss our implementation and in section 5 our validation. In section 6 we give conclusions.

2 RELATED WORK AND PRELIMINARIES

In this section we first give an overview of approaches to shape generation, and then expand on the shape language and mapping language we use in this paper.

2.1 Related work on shape generation

Shapes can express different types of validation constraints, ranging from simple data checks ("does this string match that pattern?") over cardinality checks for given properties ("does this node have at least n times this predicate?") to composition of shapes by logical connectors ("does this node match this shape and/or/xor that shape?"). The two notable RDF shape languages are ShEx [20] and the W3C-recommended SHACL [12], the latter we discuss in more detail later in this section.

Shapes can be generated manually or derived automatically. The automatic approaches can be further subdivided into approaches that derive shapes from ontologies or RDF graphs. Shapes are often generated manually, however, writing shapes by hand is costly and error-prone [6]. Editors for writing shapes, e.g., [3, 15], make shape generation less error-prone, but the process is still costly.

In previous works, automated approaches emerged as well where ontologies and the RDF graph itself were used to generate shapes to validate a graph. Approaches that generate shapes from ontologies[1,2] [5, 6, 19] are not dependent on data size, but they cannot derive any constraint beyond what the ontology already covers. Deriving shapes from an RDF graph[2] [2, 9, 18, 21, 23] is possible for any RDF graph, but the computational cost is at least linear in the data size [4]. RDF graph-based approaches suggest shapes based on available instance data and typically rely on confidence thresholds or a "human in the loop" [2] to add credence to these suggestions.

2.2 Preliminaries

In this subsection we summarize the aspects of RML and SHACL that are relevant to this paper.

RML. There are other mapping languages that generate RDF graphs from heterogeneous sources: xR2RML [17], ShExML [10], and SPARQL-Generate [13], but we consider RML because it is the most broadly used. RML [8] generates RDF graphs from heterogeneous data sources, relying on triples maps, subject maps, predicate-object maps, term maps, references and iterators.

A *triples map* (fig. 1/listing 3) describes how triples are generated from a data source by arranging a set of term maps in subject, predicate and object positions. A triples map has one *subject map* (fig. 1, ❶/listing 3, lines 2-4), a term map in the subject position, and multiple *predicate-object maps* (fig. 1, ❷/listing 3, lines 5-7), each consisting of two term maps: one in the predicate (the predicate map) and one in the object position (the object map).

A *term map* (fig. 1, ❶/listing 3, lines 2-4) describes how RDF terms are generated from source data: the node kind of the RDF term (literal, blank node or IRI), its URI template, the data type (fig. 1, ❸/listing 3, line 13) or language tag of the literals, and more. A class can also be declared for term maps in the subject position: all subjects generated by that term map have the declared class (fig. 1, ❹/listing 3, line 3). One special type of term map is used to link between triples maps: the *referencing object map* (fig. 3, ❶/listing 5, lines 13-14). If this type of object map is used, a *parent triples map* is specified, and the parent triples map's subjects are generated as the referencing object map's objects.

Term maps use *references* (fig. 1, ❺/listing 3, line 7) to refer to specific pieces of source data, for example a reference might be a column name or attribute name. References are scoped to iterations that are defined by an iterator. The *iterator* defines how a data source is iterated through, for example row-based iteration for tables, or iteration through a given array for JSON sources.

The example mapping rules in listing 3 are written for the CSV source in listing 1, and the result of these rules is the RDF graph in listing 2. The same mapping rules are given in fig. 1 in the MapVOWL visual notation [11].

SHACL. SHACL is the W3C-recommended language for expressing constraints on RDF data [12]. To express constraints on an RDF graph, SHACL uses the concepts of a data graph, a shapes graph, shapes, targets and constraint components.

The *data graph* is the RDF graph under validation, it can be, for example, a named graph in a SPARQL endpoint or a turtle file. An example data graph is shown in listing 2. The *shapes graph* contains the constraints the data graph will be validated against, in the form of a set of shapes. An example shapes graph is shown in listing 4 and its visual representation using the ShapeVOWL visual notation [15] is shown in fig. 2. *Shapes* are RDF descriptions of constraints on the data graph, using targets and constraint components. Figure 2/listing 4 contains exactly one shape: `:personShape`.

A *target* of a shape describes which nodes in the data graph will be validated by a shape. One type of target is class-based targets, stating that all instances of a given class will be validated by a shape. In our example, `:personShape` has all instances of the class `:Person` as target (fig. 2, ❶/listing 4, line 2). The other target types

[1] https://www.topquadrant.com/products/topbraid-composer/
[2] https://pypi.org/project/shaclgen/

in SHACL core state that all subjects or objects of a given property are targeted by a shape, or that specific given nodes are targeted by a shape. *Constraint components* describe the conditions against which nodes targeted by a shape are validated. There are more than 25 different types of constraint components in SHACL core and they describe various conditions such as checking whether nodes satisfy a string pattern (fig. 2, ❷/listing 4, line 5). One important type of constraint components are property constraint components (fig. 2, ❸/listing 4, lines 6-7): these check whether nodes reachable through a path (fig. 2, ❹/listing 4, line 6) conform to a shape.

3 GENERATING SHAPES FROM MAPPINGS

In this section, we propose the generation of shapes from mapping rules in two parts, first we explain how to generate a SHACL shape from a single [R2]RML triples map, using a set of [R2]RML-to-SHACL correspondences, and then we generalize this approach to generating shapes for multiple triples maps, requiring the merging and interlinking of shapes generated for single triples maps.

3.1 Shapes from one triples map

Our approach generates SHACL shapes for the RDF graph that is generated from a set of [R2]RML mapping rules. These shapes are generated considering, at this stage, only the information available in the mapping rules. The two steps to generating shapes from mapping rules are: (i) choose appropriate **targets**, and for each target, (ii), generate **constraint components**.

Choosing targets. Our approach generates class-based shape targets, because it is the most commonly used targeting approach. Statistics collected from real shapes show that class targets are used in 83% of cases, while subjects-of (`sh:targetSubjectsOf`), objects-of (`sh:targetObjectsOf`) and node targets (`sh:targetNode`) occur only 14%, 2% and 1% of cases, respectively [14].

Our approach determines all classes declared in the mapping rules and for each class generates a shape with that class as target. In our example, the class target (fig. 2, ❶/listing 4, line 2) is derived from the `:Person` class declaration in the mapping rules (fig. 1, ❹/listing 3, line 3). As stating subjects’ classes with `rr:class` or with a predicate-object map with `rdfs:type` as predicate is not required, a class can also be inferred from the domain of the predicates used in predicate maps. For example, if in fig. 1/listing 3 the `:Person` class were not declared explicitly, the target in fig. 2/listing 4 could still be inferred from the domain of `:firstName`.

The other three SHACL core targeting approaches: node targets, subject-of targets, and object-of targets are also less preferred for the following reasons: node targets are applicable when the nodes in the output are known, because they are constant or because the targets are specified after executing the mapping rules. The latter does not fit in our mappings-based approach of shape generation. Subject-of and object-of targets are not preferred in our case, because it can not be known if a predicate will be present in the output graph. For instance, the generation of the predicate or object could rely on a reference which might be null and then no RDF triple is generated.

Generating constraint components. For each class target, we add constraint components to its generated shape relying on the correspondences between [R2]RML and SHACL terms in table 1. For

```
id;firstName;lastName;age
101;Jane;Doe;25
```

Listing 1: Example non-RDF data source

```
:people/101 a :Person ; :age 25 ;
  :firstName "Jane" ;   :lastName "Doe" .
```

Listing 2: Example RDF graph

```
1  :personTriplesMap a rr:TriplesMap ;
2    rr:subjectMap [
3      rr:class :Person ;
4      rr:template "http://example.com/people/{id}" ] ;
5    rr:predicateObjectMap [
6      rr:predicate :firstName ;
7      rr:objectMap [ rml:reference "firstName" ] ] ;
8    rr:predicateObjectMap [
9      rr:predicate :lastName ;
10     rr:objectMap [ rml:reference "lastName" ] ] ;
11   rr:predicateObjectMap [
12     rr:predicate :age ;
13     rr:objectMap [ rr:datatype xsd:integer ;
14                    rml:reference "age" ] ] .
```

Listing 3: Example RML mapping rules

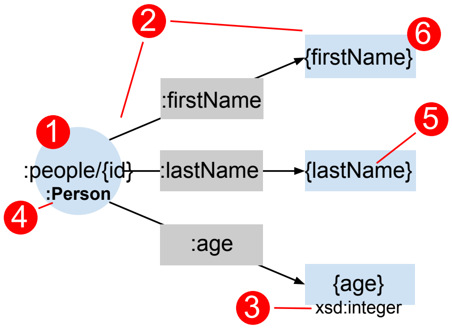

Figure 1: Example RML mapping rules (MapVOWL)

```
1  :personShape a sh:NodeShape ;
2    sh:targetClass :Person ;
3    sh:class :Person ;
4    sh:nodeKind sh:IRI ;
5    sh:pattern "http://example.com/people/.*" ;
6    sh:property [ sh:path :firstName ;
7                  sh:nodeKind sh:Literal ] ;
8    sh:property [ sh:path :lastName ;
9                  sh:nodeKind sh:Literal ] ;
10   sh:property [ sh:path :age ;
11                 sh:nodeKind sh:Literal ;
12                 sh:datatype xsd:integer ] .
```

Listing 4: Example SHACL shape

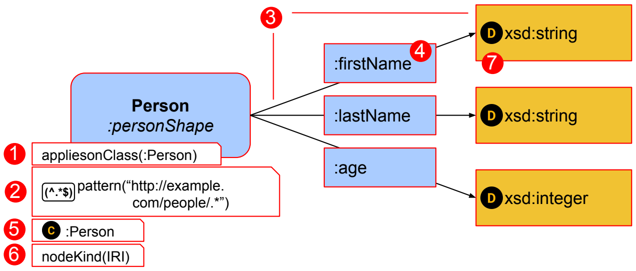

Figure 2: Example SHACL shape (ShapeVOWL)

[R2]RML	SHACL
rr:subjectMap, rr:SubjectMap	sh:NodeShape
rr:predicateObjectMap, rr:PredicateObjectMap	sh:property, sh:PropertyShape
rr:class	sh:class, sh:targetClass
rr:predicate	sh:path
rr:referencingObjectMap	sh:node
rr:termType	sh:nodeKind
rr:datatype	sh:datatype
rr:language	sh:languageIn
rr:constant	sh:in
rr:template	sh:pattern

Table 1: [R2]RML terms and corresponding SHACL terms.

```
:otherPersonTriplesMap a rr:TriplesMap ;
  rr:subjectMap [
    rr:class :Person ;
    rr:template "http://example.com/people/{id}" ] ;
  rr:predicateObjectMap [
    rr:predicate :firstName ;
    rr:objectMap [ rml:reference "firstName" ] ] ;
  rr:predicateObjectMap [
    rr:predicate :lastName ;
    rr:objectMap [ rml:reference "lastName" ] ] ;
  rr:predicateObjectMap [
    rr:predicate :country ;
    rr:objectMap [
      rr:parentTriplesMap :countryTriplesMap ] ] .
```

Listing 5: Extended example RML mapping rules

each triples map that generates entities of a class, we derive constraint components based on the properties of the triples map's subject map and predicate-object maps. For the subject map, constraint components are added directly to the class's shape, since the generated subjects have the class that this shape validates. For example, in fig. 2/listing 4 the sh:class (⑤/line 3), sh:nodeKind (⑥/line 4) and sh:pattern (②/line 5) are derived from the subject map's properties (fig. 1, ①/listing 3, lines 2-4).

For the predicate-object maps (fig. 1, ②/listing 3, lines 5-7), property constraint components are added to the class's shape (fig. 2, ③/listing 4, lines 6-7), as these constraint components can validate the predicates and objects generated by the predicate-object map. These property constraint components have a path that is derived from the predicate specified in the predicate-object map. For example the path :firstName (fig. 2, ④/listing 4, line 6) is derived from the predicate map with the same value (fig. 1, ⑥/listing 3, line 6). The property constraint components check nodes reachable through that path, so the checks performed by the property constraint component are derived from the predicate-object map's object maps, again following table 1. For example, the sh:nodeKind constraint component (fig. 2, ④/listing 4, line 7) is derived from the object map generating literals (fig. 1, ⑥/listing 3, line 7).

In the above, we treat the entire output of [R2]RML mapping rules as one data graph. However, [R2]RML can generate multiple data graphs, either by (i) generating multiple named graphs with rr:graph(Map) or (ii) writing to multiple output locations with rmlt:target [22]. Yet, our approach can also be restricted to, e.g., only generate shapes for one named graph or output target.

3.2 Shapes from multiple triples maps

[R2]RML mapping rules are used to integrate multiple sources into one RDF graph, and these sources can have overlapping or complementary data. Both cases require mapping rules for which the shape generation approach explained so far (section 3.1) is incomplete and we explain a more complete approach in this section.

Sources with *overlapping data* contain the same types of entities (e.g., people) and similar attributes of those entities. These data sources typically contain overlapping sets of attributes of the entities: some attributes are shared between sources (e.g., all sources contain people's names and identifier) while other attributes are not (e.g., only certain sources know people's age or country of birth). Following the approach in section 3.1, we would generate different shapes for each data source, even though the shapes would be redundant with each other and could be merged.

Sources with *complementary data* contain similar entities but different attributes. One source refers to a type of entities, and another source contains more information about those entities. For example, one source contains information about people and a code for their birth country, and another source contains more information about countries. If complementary data sources' RDF representation is validated with shapes, the shapes for the sources should be linked just like the sources; in our example the person shape should refer to the country shape.

In the remaining, we explain how these cases are handled.

Sources with overlapping data. [R2]RML mapping rules can integrate multiple sources with overlapping data by generating entities with the same class from multiple sources. An example is shown in fig. 1/listing 3 and fig. 3/listing 5: both triples maps create instances of the :Person class with the properties :firstName and :lastName. One triples map additionally adds the :age property, and the other :country.

Our approach generates one shape for all triples maps that generate entities of the same class. This is achieved by generating a shape which checks the properties that all instances of the class should have (in our example, :firstName and :lastName) and optionally

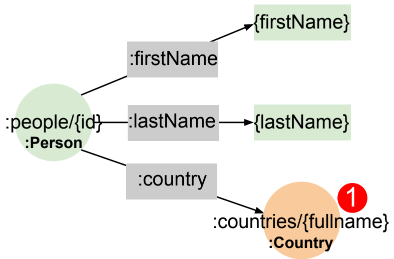

Figure 3: Extended example RML mapping rules (MapVOWL)

checks the properties that only some of the class instances should have (in our example, `:age` or `:country`). Determining which properties all of a class's instances have, is done by looking at all triples maps that generate instances of this class, and determining their shared subject maps and predicate-object maps. Once these are determined, a shape is generated for this class much like in section 3.1, except that the *non-shared* properties are added as a disjunction (`sh:or`), as shown on fig. 4, ❶/listing 6, lines 10-16.

Complementary data. Sources with complementary data can be integrated into a knowledge graph by using [R2]RML mapping rules with a referencing object map. If a referencing object map is used, the terms generated by that object map are the subjects generated by its parent triples map, which might have a different source. For example, the object map in fig. 3, ❶/listing 5, line 14 generates IRIs for `:Country` instances from another source than the source used to create `:Person` instances; such a scenario is likely if the person source refers to countries by a code, while IRIs using the countries' full names are desired. Since the terms generated by a referencing object map conform to the shape generated for the parent triples map, the constraint components generated for referencing object maps likewise refer to the shape generated for the parent triples map. This is indicated in fig. 4, ❷/listing 6, line 16: using `sh:node`, the objects of the `:country` predicate are validated by `:countryShape`, the shape generated for the `:Country` class.

```
1  :personShape a sh:NodeShape ;
2    sh:targetClass :Person ;
3    sh:class :Person ;
4    sh:nodeKind sh:IRI ;
5    sh:pattern "http://example.com/.*" ;
6    sh:property [ sh:path :firstName ;
7                  sh:nodeKind sh:Literal ] ;
8    sh:property [ sh:path :lastName ;
9                  sh:nodeKind sh:Literal ] ;
10   sh:or ( [ sh:property [
11             sh:path :age ;
12             sh:nodeKind sh:Literal ;
13             sh:datatype xsd:integer ] ]
14           [ sh:property [
15             sh:path :country ;
16             sh:node :countryShape ] ] ) .
```

Listing 6: Extended example SHACL shapes

```
   Data: Set of mapping rules $M$
   Result: Shapes graph $S$
1  $S \leftarrow$ empty graph;
2  for each class $c$ declared in $M$ do
3      $s \leftarrow$ empty node shape with target class $c$;
4      $o \leftarrow$ sh:or with empty list parameter $L$;
5      $T \leftarrow$ all triples maps declaring $c$;
6      for each triples map $tm$ in $T$ do
7          $sub \leftarrow$ constraints for $tm$'s subject map (table 1);
8          if $sub$ is equal for all subject maps in $T$ then
9              $s \leftarrow s \cup sub$
10         else $L \leftarrow L$ ++ $sub$ ;
11         for each predicate-object map $po$ of $tm$ do
12             $pcc \leftarrow$ property constraint component with $po$'s
                 predicate as path and with constraint
                 components for $po$'s object map (table 1);
13             if $pcc$ is shared between all triples maps in $T$ then
14                 $s \leftarrow s \cup cc$
15             else $L \leftarrow L$ ++ $pcc$ ;
16     if $L$ is not empty then $s \leftarrow s \cup o$ ;
17     $S \leftarrow S \cup s$;
18     Return $S$
```

Algorithm 1: Pseudocode for generating SHACL shapes from [R2]RML mapping rules

4 IMPLEMENTATION

We implemented our proposed approach to generating SHACL shapes from [R2]RML mapping rules. We present the implementation's pseudocode and then discuss the implementation itself.

Algorithm. The following pseudocode (alg. 1) is an algorithm for the approach we proposed in section 3. For each class declared in the mapping rules (loop on lines 2-15) constraint components are generated that validate the properties of the class instances (lines 7, 12). The shared and unique properties are determined (lines 8, 13) and the shared constraint component are added to the class's shape (lines 9, 14) while the unique properties are added to a disjunction (lines 11, 15) that in its turn is added to the class's shape (line 16).

Python implementation. A proof of concept implementation under the permissive MIT license is available[3]. It is written in Python using the RDFLib library.[4] The implementation follows the approach explained in section 3 and formalized in alg. 1, without merging shapes from multiple triples maps. The implementation also has partial support for inferring target classes from predicates' domains for certain cases, as explained in section 3.1.

5 VALIDATION

We assessed three aspects of our approach to generating shapes from mapping rules: (i) the extent to which it covers the different aspects of the RML specification, (ii) whether it generates shapes that

[3] https://github.com/RMLio/RML2SHACL/tree/kcap21
[4] https://github.com/RDFLib/rdflib

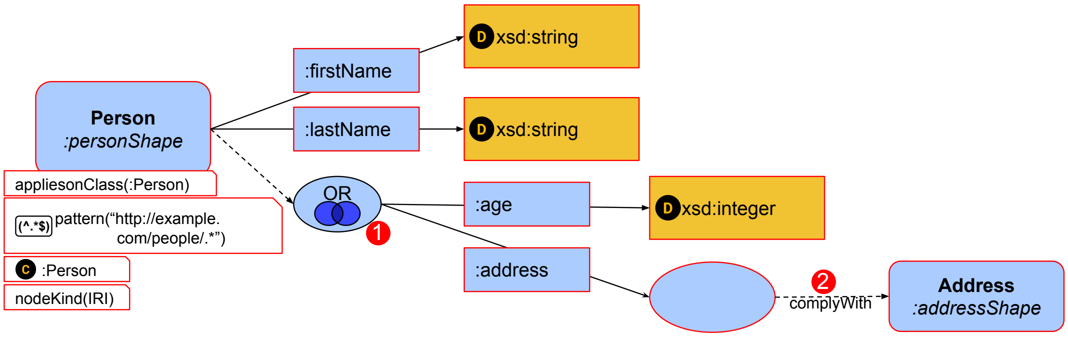

Figure 4: Extended example SHACL shapes (ShapeVOWL)

are comparable with shapes generated by state of the art ontology-based methods and (iii) at what data size our mapping-based approach is less costly than data graph-based approaches.

5.1 RML test cases

To verify if our approach generates meaningful shapes for a large range of RML mapping rules, we validated our implementation relying on the RML test cases.

Methodology. We applied our implementation on the mapping rules of each of the RML test cases[5]. These test cases are designed by the KG-Construct community group to (among other goals) have high coverage of the different RML constructs. There are more than 300 test cases and they contain mapping rules with many different combinations of RML constructs, such as term maps, source types, language and data type maps. For each test case, we validate the shapes graphs returned by our implementation using the SHACL-SHACL shapes.[6] SHACL-SHACL is itself a shapes graph that can validate whether shapes graphs are well-formed.

Results. The results of these validations are positive: for all RML test cases with well-formed mapping rules, well-formed shapes graphs are generated. These results indicate that our implementation can cover a large range of different mapping rules and, thus, of the RML specification. The shapes graphs generated for each test case are published online, along with the code to run them.[7]

5.2 Comparison with ontology-based tools

Mapping-based and ontology-based approaches are similar, though not identical: both require no user input and are independent of the data graph's size. Because of this similarity, it is interesting to investigate how shapes generated by both approaches are similar, where their shapes differ and how they can be combined, if desirable.

Ontology-based approaches for shape generation take OWL, RDFS and XSD restrictions as input. An OWL ontology for our running example is given in fig. 5 and shown in listing 7 in the VOWL notation [16]: like the shape (fig. 2/listing 4) and the mapping rules (fig. 1/listing 3) shown earlier, the ontology encodes that `:Person` instances have first and last names that are plain literals, and age that is an integer. One remarkable difference with the shape and mapping rules is that fig. 5/listing 7 does not contain an URI template or pattern, as ontologies have no way to encode this.

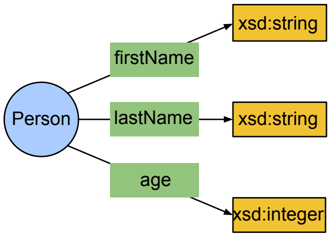

Figure 5: Example OWL ontology (VOWL)

Methodology. As representative of the ontology-based approaches, we chose to compare with Astrea [6], a state of the art tool that generates shapes from ontologies and requires no user input. Astrea publishes both its code[8] as well as a catalogue of generated shapes from well-known ontologies online[9]. For each ontology in Astrea's catalogue, we generated mapping rules using a third party OWL-to-RML tool[10]. From these generated mapping rules, we generate shapes using RML2SHACL's implementation. Then we observe the similarities and difference of shapes generated by the two tools for the same class. The shapes generated by both tools and the used mapping rules are published along with the code[11].

Results. We observe that both tools generate shapes with similar structure, while the mapping-rules-based approach better supports validating RDF properties related to data values (like URI patterns and node kinds), and the ontology-based approach better supports validating schema properties (like class disjointness).

Both approaches generate similar shapes for the same class: both approaches generate shapes that validate mainly node kinds, data types, and property paths. Samples of shapes generated by both tools are shown in listings 8 and 9: these shapes are generated for the `:Item` class in the OpenADR ontology[12] and both shapes validate

[5] https://github.com/kg-construct/rml-test-cases
[6] https://www.w3.org/TR/shacl/#shacl-shacl
[7] https://github.com/RMLio/RML2SHACL/tree/kcap21/shapes
[8] https://github.com/oeg-upm/astrea
[9] https://astrea.linkeddata.es/catalogue.html
[10] https://github.com/oeg-dataintegration/owl2rml
[11] https://github.com/RMLio/RML2SHACL/tree/kcap21/shapes/astrea_test
[12] https://w3id.org/def/openadr

```
:Person rdf:type owl:Class .
:firstName rdf:type owl:DatatypeProperty ;
  rdfs:domain :Person ;
  rdfs:range xsd:string .
:lastName rdf:type owl:DatatypeProperty ;
  rdfs:domain :Person ;
  rdfs:range xsd:string .
:age rdf:type owl:DatatypeProperty ;
  rdfs:domain :Person ;
  rdfs:range xsd:integer .
```

Listing 7: Example OWL ontology

```
1  rml2shacl:triplesMapItem_000/shape
2    sh:nodeKind sh:IRI ;
3    sh:pattern "http://www.example.com/.*" ;
4    sh:property
5      [ sh:nodeKind sh:Literal ;
6        sh:path :hasItemDescription ],
7      [ sh:nodeKind sh:Literal ;
8        sh:path :hasScaleCode ] ;
9      sh:targetClass :Item .
```

Listing 8: Shape generated by RML2SHACL

```
1   astrea:eb103427682b59681883e1aa14df7b2e
2     sh:nodeKind sh:IRI ;
3     sh:property
4       astrea:627e8be20efe1fd3c04d5e3b9876377b ,
5       astrea:841377b51482efab5bbd280e2513f34c ;
6     sh:targetClass :Item .
7   astrea:841377b51482efab5bbd280e2513f34c
8     sh:datatype xsd:string ;
9     sh:nodeKind sh:Literal ;
10    sh:path :hasItemDescription ;
11    sh:pattern ".*" .
12  astrea:627e8be20efe1fd3c04d5e3b9876377b
13    sh:datatype xsd:string ;
14    sh:nodeKind sh:Literal ;
15    sh:path :hasScaleCode ;
16    sh:pattern ".*" .
```

Listing 9: Shape generated by Astrea

whether `:Item` instances are IRIs that have `:hasItemDescription` and `:hasScaleCode` properties whose objects are literals.

The tools can generate shapes that are different in style but equal in meaning. For example, for predicates with a given class as range, RML2SHACL refers to the range class's shape explicitly by using `sh:node`, while Astrea's shapes will check simply if the objects have the range class with `sh:class` (which is enough: instances of that class are targeted and validated by that class's own shape). There are other, smaller, stylistic differences: use of blank nodes vs. IRIs for shapes (e.g., listing 8, lines 5-7 vs. listing 9, lines 7-11), and in- or excluding names, labels and descriptions of shapes.

RML2SHACL can generate more specific constraint components than Astrea in two cases: URI patterns and node kinds. Mapping rules use URI templates to generate URIs and from these templates, pattern constraint components can be derived (e.g., listing 8, line 4); such information is not available in ontologies. Likewise for node kinds, such information is not always encoded in ontologies, so ontology-based tools cannot determine whether nodes are literals or IRIs or blank nodes, meanwhile mapping rules encode how nodes are generated, so typically their node kinds *are* encoded in mapping rules. This is reflected in generated shapes where, for the same nodes, Astrea's shape has `sh:LiteralOrIRI` as node kind, while RML2SHACL has the more specific `sh:IRI`.

Shapes generated by Astrea sometimes contain more constraint components than RML2SHACL's shapes. One such occasion is when the ontology contains `owl:disjointWith` statements for classes, which Astrea can translate to equivalent `sh:not` constraint components. We also noticed Astrea cleverly generates certain constraint components based on latent information not necessarily taken from the ontology. These constraint components therefore could also be added to RML2SHACL's implementation. Such derived constraint components are based on facts like these: literals with a datatype have a corresponding string pattern (e.g., an `xsd:integer` is a string of numbers) (used in listing 9 lines 11, 16), and literals are strings unless stated otherwise (used in listing 9 lines 9, 13).

5.3 Comparison with RDF graph-based tools

To verify the merits of an approach which is not dependent on the data-size, we compared our approach against approaches that generate shapes from an RDF graph while measuring resource use.

Methodology. From the open source RDF-graph-based shape generation tools: ShaclGen[13], RDF2Graph [23] and RDFShapeInduction [18], we chose to compare against ShaclGen, as the other two were either deprecated[14] (no updates in 6 years and errors on running) or contained no installation instructions[15]. We generated RDF graphs of different sizes using the BSBM data generator: from 500 products (graph size 45MB) up to 5000 products (graph size 440MB). We then ran SHACLGen five times on each generated RDF graph, measuring the average execution time per size. We also ran RML2SHACL on the BSBM RML mapping rules five times while measuring time. Our experimental set-ups are publicly available[16].

Results. Our experiment confirms that the resource cost of RDF-graph-based tools for shape generation increases at least linearly with the RDF graph's size. We found that once the RDF graph's size exceeds 90MB (1000 products), execution time of these tools is higher than the execution time of mapping-based tools, which remains constant relative to data size, as shown in fig. 6. Constant execution time comes at a cost: the BSBM data relies for a part on dynamically generated classes. These classes are invisible to mapping-based approaches, but data-based approaches can (and ShaclGen does) generate shapes for these classes.

6 CONCLUSION

We present a novel approach for making RDF shapes for RDF graphs generated with mapping rules. Our approach is automatic, i.e., needs no user input, and does not depend on the graphs' size.

[13] https://github.com/uwlib-cams/shaclgen
[14] https://github.com/jessevdam/RDF2Graph
[15] https://github.com/rifat963/RDFShapeInduction
[16] https://github.com/RMLio/RML2SHACL/tree/kcap21/

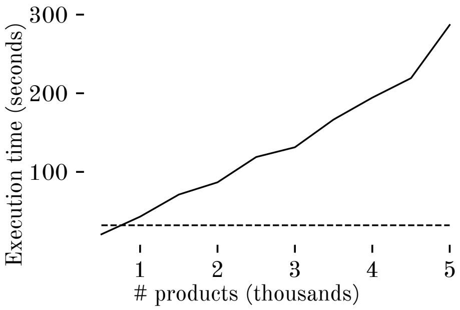

Figure 6: Resource use of data graph-based tools (full line) increases with the data graph size, while that of mapping-based tools (dashed line) remains constant.

Our experiments confirm that our mapping-based approach generates shapes from a large range of mappings, similar to ontology-based approaches, with slightly more constraint components that are data value-based and slightly less constraint components that are schema-based, and faster than data-graph based approach for data sizes of 90MB and higher.

RML2SHACL brings shape generation to a new class of use cases: those where no manual intervention or excessive resource use is desired and where either no ontology is available, or different constraint components are desired than those encoded in ontologies.

In the future we will investigate generalizing our approach by also generating RDF shapes encoded in ShEx [20].

Since the shapes generated by our mapping-based approach and by ontology-based approaches are complementary, we will research further on how to combine the two approaches. Our approach could be enhanced by taking more information (besides domains of predicates) from ontologies, e.g., cardinalities or data ranges. Mapping rules can use a selection of terms from different ontologies, so generating shapes from mapping rules makes it possible to integrating and selecting the relevant information from different ontologies.

The relation between mapping-based and RDF graph-based shape generation also deserves further investigation. From the mapping rules "skeleton shapes" could be generated which will necessarily validate, then data analysis could suggest more constraint components to add to these skeletons. Data analysis might be easier on non-RDF source data than on RDF graphs, since the source data typically is more rigidly structured (e.g., tabular), might have more available analysis tooling, or might even have a pre-set schema.

ACKNOWLEDGMENTS

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

REFERENCES

[1] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. 2012. *A Direct Mapping of Relational Data to RDF.* Recommendation. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdb-direct-mapping/

[2] Iovka Boneva, Jérémie Dusart, Daniel Fernández-Álvarez, and José Emilio Emilio Labra Gayo. 2019. Semi Automatic Construction of ShEx and SHACL Schemas. *CoRR* abs/1907.10603 (2019). arXiv:1907.10603

[3] Iovka Boneva, Jérémie Dusart, Daniel Fernández Alvarez, and Jose Emilio Labra Gayo. 2019. Shape Designer for ShEx and SHACL Constraints. ISWC 2019 - 18th International Semantic Web Conference. In *ISWC 2019 Satellites.*

[4] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing semantic graphs: a survey. *The VLDB journal* 28, 3 (2019), 295–327.

[5] Kārlis Čerāns, Jūlija Ovčiņņikova, Uldis Bojārs, Mikus Grasmanis, Lelde Lāce, and Aiga Romāne. 2021. Schema-Backed Visual Queries over Europeana and Other Linked Data Resources. In *The Semantic Web: ESWC 2021 Satellite Events.* Cham, 82–87.

[6] Andrea Cimmino, Alba Fernández-Izquierdo, and Raúl García-Castro. 2020. Astrea: Automatic Generation of SHACL Shapes from Ontologies. *The Semantic Web* 12123 (2020), 497–513.

[7] Souripriya Das, Seema Sundara, and Richard Cyganiak. 2012. *R2RML: RDB to RDF Mapping Language.* Working Group Recommendation. World Wide Web Consortium (W3C). http://www.w3.org/TR/r2rml/

[8] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. 2014. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, Vol. 1184.

[9] Daniel Fernández-Álvarez, H. García-González, Johannes Frey, S. Hellmann, and Jose Emilio Labra Gayo. 2018. Inference of Latent Shape Expressions Associated to DBpedia Ontology. In *International Semantic Web Conference.*

[10] Herminio García-González, Iovka Boneva, Sławek Staworko, José Emilio Labra-Gayo, and Juan Manuel Cueva Lovelle. 2020. ShExML: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Computer Science* 6 (2020), e318.

[11] Pieter Heyvaert, Anastasia Dimou, Ben De Meester, Tom Seymoens, Aron-Levi Herregodts, Ruben Verborgh, Dimitrie Schuurman, and Erik Mannens. 2018. Specification and implementation of mapping rule visualization and editing: MapVOWL and the RMLEditor. *Web Semantics: Science, Services and Agents on the World Wide Web* 49 (March 2018), 31–50.

[12] Dimitris Kontokostas and Holger Knublauch. 2017. *Shapes Constraint Language (SHACL).* W3C Recommendation. W3C. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[13] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. 2017. A SPARQL Extension for Generating RDF from Heterogeneous Formats. In *The Semantic Web 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings.* Portoroz, Slovenia, 35–50.

[14] Sven Lieber, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. 2020. Statistics about Data Shape Use in RDF Data. In *Proceedings of the 19th International Semantic Web Conference: Posters, Demos, and Industry Tracks*, Vol. 2721. 330–335.

[15] Sven Lieber, Ben De Meester, Pieter Heyvaert, Femke Brückmann, Ruben Wambacq, Erik Mannens, Ruben Verborgh, and Anastasia Dimou. 2021. Visual Notations for Viewing RDF Constraints with UnSHACLed [to be published]. *Semantic Web Journal* (2021).

[16] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. 2016. Visualizing ontologies with VOWL. 7 (2016), 399–419. https://doi.org/10.3233/sw-150200

[17] Franck Michel, Loïc Djimenou, Catherine Faron-Zucker, and Johan Montagnat. 2015. Translation of Heterogeneous Databases into RDF, and Application to the Construction of a SKOS Taxonomical Reference. In *International Conference on Web Information Systems and Technologies.* 275–296.

[18] Nandana Mihindukulasooriya, Mohammad Rifat Ahmmad Rashid, Giuseppe Rizzo, Raul Garcia-Castro, Oscar Corcho, and Marco Torchiano. 2017. RDF Shape Induction using Knowledge Base Profiling. In *Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing.*

[19] Jędrzej Potoniec, Piotr Jakubowski, and Agnieszka Ławrynowicz. 2017. Swift linked data miner: Mining OWL 2 EL class expressions directly from online RDF datasets. *Journal of Web Semantics* 46 (2017), 31–50.

[20] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. 2014. Shape expressions: an RDF validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems.* New York, NY, United States, 32–40.

[21] Blerina Spahiu, A. Maurino, and M. Palmonari. 2018. Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL. In *Workshop on Ontology Design Patterns (WOP) at ISWC (Best Workshop Papers).*

[22] Dylan Van Assche, Gerald Haesendonck, Gertjan De Mulder, Thomas Delva, Pieter Heyvaert, Ben De Meester, and Anastasia Dimou. 2021. Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation. In *Web Engineering (Lecture Notes in Computer Science, Vol. 12706).* Springer, Cham.

[23] Jesse CJ van Dam, Jasper J Koehorst, Peter J Schaap, Vitor Ap Martins Dos Santos, and Maria Suarez-Diez. 2015. RDF2Graph a tool to recover, understand and validate the ontology of an RDF resource. *Journal of biomedical semantics* 6, 1 (2015), 1–12.