SCOOP all the Constraints' Flavours for your Knowledge Graph

Xuemin Duan[1][0000−0002−3256−8341], David Chaves-Fraga[2,1][0000−0003−3236−2789], Olivier Derom[1], and Anastasia Dimou[1][0000−0003−2138−7972]

[1] KU Leuven – Flanders Make@KULeuven – Leuven.AI, Belgium
{xuemin.duan,anastasia.dimou}@kuleuven.be & olivierderom@gmail.com
[2] Grupo de Sistemas Intelixentes, Universidade de Santiago de Compostela, Spain
david.chaves@usc.es

Abstract. Creating SHACL shapes for the validation of RDF graphs is a non-trivial endeavor. Automated shape extraction systems typically derive SHACL shapes from RDF graphs, and thus, their effectiveness is inherently influenced by the size and complexity of the RDF graph. However, these systems often overlook the constraints imposed by individual artifacts, although RDF graphs are often constructed by applying ontology terms to heterogeneous data. Only a few systems extract SHACL shapes from either the data schema or the ontology, leading, in either case, to limited or incomplete constraints. We propose SCOOP, a framework that exploits all artifacts associated with the construction of an RDF graph, i.e. data schemas, ontologies, and mapping rules, and integrates the SHACL shapes extracted from each artifact into a unified shapes graph. We applied our approach to real-world use cases and experimental results showed that SCOOP outperforms systems that extract SHACL shapes from RDF graphs, generating more than double the types of constraints than those systems, and effectively identifying missing and erroneous RDF triples during the validation process.
Resource type: Software Framework — **License**: Apache-2.0
DOI: `https://doi.org/10.5281/zenodo.10280346`
URL: `https://github.com/dtai-kg/SCOOP`

Keywords: SHACL · Shape integration · Knowledge graph validation.

1 Introduction

Creating SHACL shapes for the validation of RDF graphs is a non-trivial endeavor. Shapes Constraint Language (SHACL) [28], the W3C recommendation for validating RDF graphs, has gained increasing popularity, which has further prompted a growing number of endeavors in automating shape extraction. However, a notable proportion of users in industry and academia still favor creating shapes manually [15,33], indicating the practical limitations of existing works.

Currently, automated shape extraction broadly falls into two categories: RDF-based and non-RDF-based systems. The cost and performance of systems extracting shapes from *RDF* graphs [18,19,20,29,34,39] are inherently contingent upon the scale and intricacy of the RDF graph, frequently leading to a limited subset of constraints [32], e.g., not covering value range (e.g., sh:minExclusive) or string-based constraints (e.g., sh:minLength). Besides, these systems also overlook the constraints imposed by an RDF graph's individual artifacts.

Non-RDF systems can be further delineated into extraction systems mining shapes from either ontologies, mappings, or raw data schemas, i.e. the artifacts from which RDF graphs are constructed. However, extracting shapes only from *ontologies* [6,24] often results in redundant shapes, as the shapes frequently encompass a broader spectrum of classes and properties than those in the target RDF graph. Extracting shapes from *raw data schemas*, e.g., SQL schema [40], XML Schema Definitions (XSD) [14,22] or JSON Schema [8], always results in the misalignment between the extracted shapes and the classes and properties within the target RDF graph [14]. Extracting shapes from *mappings* [11] covers only a limited set of constraints [11]. This singular input source often results in limited or incomplete constraints. To date, no research has been undertaken to integrate shapes from multiple relevant sources into a unified shapes graph, alleviating the drawbacks of individual systems and bolstering overall robustness.

We present **SCOOP**, a framework that exploits all artifacts associated with the construction of an RDF graph: data **SC**hemas, **O**nt**O**logies, and ma**P**pings. SCOOP integrates the SHACL shapes extracted from each artifact into a unified shapes graph. To achieve this, SCOOP comprises three modules: (i) *post-adjustment* to align schema-driven shapes with the target RDF graph; (ii) *equivalences identification* to align shapes from diverse sources, and (iii) *integration and inconsistencies resolution* to prevent unsatisfiable shapes. We implement SCOOP as an open-source system[1] which incorporates RML2SHACL [10,11], Astrea [5,7], and XSD2SHACL [13,14] to integrate shapes from mappings in RML [12,26], ontologies in OWL [3], and raw data schemas in XSD [17].

We compare SCOOP with state-of-the-art systems in a real-world use case. The performance experiments show that SCOOP has a significantly lower maximum memory usage compared to the RDF-based system. Addtionally, its running time is independent of the data size and faster than the RDF-based system when the RDF graph exceeds 3.30 GB in size. The coverage experiments reveal that SCOOP excels in generating accurate shapes to target classes and properties, outperforming state-of-the-art systems, and generates more than twice the types of constraints compared to RDF-based systems. SCOOP is an extensible framework that can seamlessly integrate more sources as additional shape extraction systems emerge, e.g., other raw data schemas or mapping languages.

The remaining paper is organized as follows: Section 2 describes related works and preliminaries; Section 3 discusses the challenges during shape integration; Section 4 describes our methodology and implementation to address these challenges. Section 5 provides the evaluation of SCOOP in real-world use cases. Section 6 concludes this paper and outlines possible future directions.

[1] `https://github.com/dtai-kg/SCOOP`

2 Preliminaries and Related Work

In this section, we introduce RDF validation (SHACL) and RDF construction (RML), and then outline related work regarding shape extraction.

2.1 Preliminaries

This subsection summarizes the concepts required to understand the paper.

KG Validation with SHACL. To validate nodes in the RDF graph, the Shapes Constraint Language (SHACL) was recommended by W3C [28]. The *shapes graph* refers to a collection of shapes containing constraints expressed in RDF format, while the *data graph* pertains to the target RDF graph undergoing validation. A *node shape* typically specifies constraints on the focus node, where the focus node represents the node in the RDF graph being validated (e.g., all instances of :Student, Fig. 2). The *focus node* can be directly designated by *target declaration* (e.g., :NS sh:targetClass :Student), indirectly specified through reference relationships (e.g., :NS sh:node :NS2), or explicitly fed to the SHACL processor with nodes to be validated. In cases where RDF graphs are to be validated automatically without explicit extra specification, shapes lacking target declarations become invalid as they do not trigger validation. A property shape primarily serves to define the property value reached from the focus node through a property path (e.g., :PS sh:path :grade). Therefore, reference relationships between shapes (e.g., :NS sh:property :PS) are crucial, as they delineate how the SHACL processor traverses from the focus node to the property value.

KG Construction with RML. To map heterogeneous data to RDF graphs, the RDF Mapping language was proposed in [12,26] generalizing R2RML [9], the W3C recommendation to construct RDF graphs from relational databases. The triples map describes how to map a data source, comprised by a logical source, a subject map, and multiple predicate-object maps. The logical source defines information about the mapped data source, including data location (e.g., "student.xml" in Fig. 2), reference formulation (e.g., ql:XPath in Fig. 2), and an optional iterator (e.g., "/student" in Fig. 2). The subject map defines how the subjects of triples are generated (e.g., "http://example.com/{@id}" in Fig. 2), and the predicate-object map defines how the corresponding predicate (e.g., :grade in Fig. 2) and object (e.g., "/grade" in Fig. 2) are generated. The reference is used to directly specify the original data source, while the template is used to build strings with the data source, which can be a single-component template or a multi-component template (e.g., "http://example.com/{@id}_{grade}").

2.2 Related Work

Prior research on SHACL shape extraction primarily focuses on deriving shapes graphs from *RDF graphs*; but also *ontologies*, *raw data schemas*, and *mapping rules*. However, no system integrates multiple shapes into a unified shape.

RDF graphs-driven. Numerous efforts were proposed to extract shapes and constraints from *RDF graphs* but the computational burden is contingent upon the scale of the RDF graphs and the covered constraints are limited [32]. **QSE** [34] mines shapes from large RDF graphs and avoids spurious shapes by calculating the support and confidence scores of extracted shapes; **SHACTOR** [35] provides a graphical user interface based on QSE. **SHACLGEN** [2] implements a query-based system, which loads the whole RDF graph in memory restricting its capacity to handle large RDF graphs. **SheXer** [19,20] mines and filters feature for constraints according to the frequency of occurrence, but the implementation lacks support for N-Quads. **ABSTAT** [39] employs a summarization model to mine shapes by summarizing triple patterns extracted from the semantic profile of RDF graphs. **ShapeInduction** [29] proposes an RDF graph profiling approach using machine learning techniques by transforming the regression problem into a classification problem. **SHACLearner** [23] utilizes the inverse open path (IOP) to infer potential constraints in shape fragments for a given target. Last, **ShapeDesigner** [4] extracts shapes using a query language.

Ontology-driven. Deriving SHACL shapes from *ontologies* is independent of data scale, yet limited to constraints exclusively defined by the ontology, and frequently derives redundant shapes as the RDF graph may only involve a subset of the ontology. **Astrea** [7] derives SHACL shapes from ontologies by executing a set of pre-defined SPARQL queries to retrieve ontology elements and derive corresponding shapes and constraints. **SHACLGEN** also supports extracting shapes from ontologies and **Harshvardhan et al.** [24,30] reuse the Ontology Design Pattern (ODP) axioms to extract equivalent SHACL shapes.

Schema-driven. Extracting SHACL shapes from *raw data schemas* eliminates the dependencies on the RDF graph scale, akin to ontology-driven systems. However, the extracted shapes resort to a default namespace, causing a misalignment with their targeted RDF graphs. For XML, **XSD2SHACL** [14] mines SHACL shapes based on a set of correspondences between XSD and SHACL, while **XMLSchema2ShEx** [22] mines ShEx shapes, but partially covers XSD.

Mapping-driven. Deriving shapes from *mapping rules* [11] captures the implied constraints within the mapping rules, but it is limited to the constraints that can be described by the mapping language. **RML2SHACL** [11] is the only system that derives SHACL shapes from RML mapping rules [12] based on a set of proposed correspondences. Its coverage is limited to constraints of a more basic nature, as RML does not offer the same scope of support for rich constraints found in SHACL, such as value range constraints (e.g., sh:minExclusive).

As opposed to the aforementioned, **SCOOP** integrates SHACL shapes extracted from the raw data schema, the ontology, and the mapping rules, taking advantage of all aspects that contributed to the construction of an RDF graph.

3 Shape Integration Challenges

Integrating SHACL shapes from mapping rules, ontologies, and raw data schemas is not straightforward, as challenges may arise during the integration process.

(C1) Misalignments. Misalignment arises inevitably during the integration of shapes extracted from raw data schemas and those extracted from ontology or mappings. The shapes derived from mappings and ontologies are swiftly applicable for validation purposes, as they inherently align with the defined classes and properties within the RDF graph (e.g., sh:targetClass of stu:Student).

However, the shapes from raw data schemas cannot be directly aligned with the SHACL shapes extracted from the mappings and ontologies. Due to the absence of classes and properties information specific to the target RDF graphs, the extracted shapes resort to default namespaces pre-defined by the extraction systems. As shown in Fig. 2, the *XSD-driven* shapes opt for ex:student as target class (instead of stu:Student as in the target RDF graph). Moreover, constraints extracted from the raw data schema may be superseded or become invalid during the RDF construction. For instance, the id originally defined as a string in *XSD-driven* shapes may be used to create an IRI according to the mappings.

(C2) Inequivalences. The basis of the shape integration lies in the consolidation of constraints encapsulated within equivalent shapes, i.e. shapes validating the same node, which cannot be solely ascertained through uniform target declarations or property paths due to the flexibility of SHACL syntax. For instance, a node shape specified with sh:targetClass :Student is deemed equivalent to a shape that defines the same target declaration. However, it may also be considered equivalent to another node shape referenced by a shape sharing the same target declaration (e.g., sh:targetClass :Student) through sh:node, even if the latter node shape lacks the same target declaration.

(C3) Inconsistencies. Integrating shapes from different sources may lead to constraint inconsistencies, potentially rendering the unsatisfiable shape, i.e., a shape that can never be conformed. Two distinct types of inconsistencies are identified: one type adheres to SHACL syntax without violation, while the other type poses a risk to the well-formedness of the resulting shape. As an illustrative scenario, consider the construction of an RDF graph from two distinct raw data sources, each is adhering to dissimilar grading systems. One source specifies a passing grade range spanning from 8 to 20 points and the student id should be IRI, while the other designates that from 60 to 100 and the id should be a string. Direct integration of constraints of sh:nodeKind sh:IRI and sh:datatype :string will lead to inconsistency, i.e. it does not violate syntax but will cause the shape to never be satisfied. The simultaneous presence of two distinct passing grade constraints sh:minInclusive and two sh:maxInclusive can be categorized as an inconsistency, which arises from the restriction that a shape incorporates at most one sh:minInclusive constraint and one sh:maxInclusive constraint.

4 SCOOP

We propose SCOOP, an open-source framework[1] designed to integrate existing shape extraction systems targeting diverse sources into a unified and comprehensive shapes graph. The current version of SCOOP supports SHACL core [28].

The integration workflow (Fig. 1) can be delineated into three principal modules: (1) **post-adjustment** of shapes extracted from raw data schema to ensure alignment with the target RDF graph (Fig. 1–1), overcoming (C1); (2) **equivalences identification** among extracted shapes from diverse sources to identify

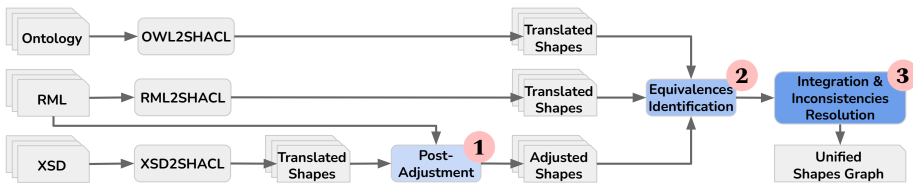


Fig. 1: The workflow of SCOOP.

equivalent shapes (Fig. 1–2), overcoming (C2); (3) **integration and inconsistencies resolution** for identifying inconsistent constraints within equivalent shapes (Fig. 1–3) to make integration decisions based-on three configurations, SCOOP-All, SCOOP-Prior, and SCOOP-Prior-R, overcoming (C3). The modules are explained in detail in Subsection 4.1, 4.2, and 4.3 respectively.

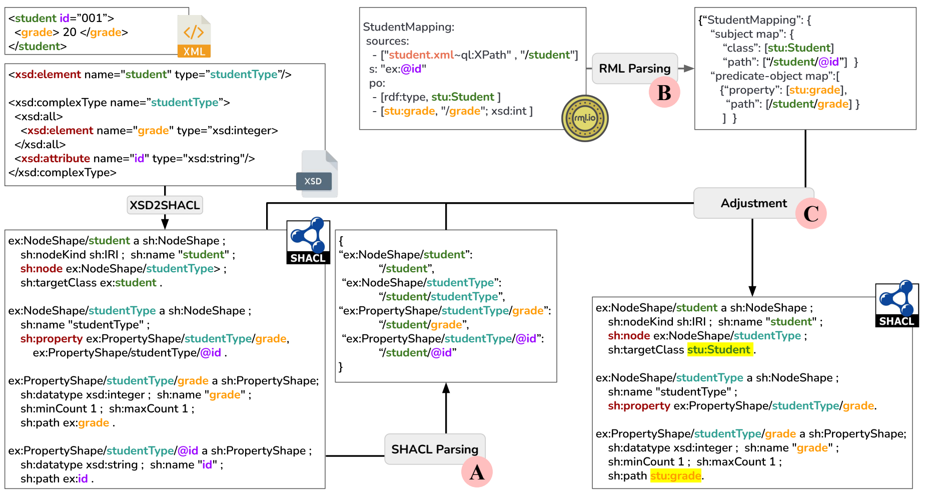
4.1 Post-Adjustment

We mitigate the misalignment between preliminary shapes extracted from raw data schemas and target RDF graphs by leveraging the RML mappings during **post-adjustment**. Given that the most mature systems to extract shapes from the raw data schema are focused on XML, we demonstrate our methodology for XML Schema but the same should hold for other schemas as well.

The post-adjustment (Fig. 2) is structured into three steps: (a) **SHACL parsing** to compute the implicit XPath within each shape (e.g., /student/grade) (Fig. 2–A); (b) **RML parsing** to extract the XPath alongside the corresponding classes and properties (e.g., the property stu:grade is related to objects constructed with the values of the elements matching the /student/grade XPath expression) (Fig. 2–B); (c) shape **adjustment** based on the outcomes of the parsing process (e.g., adjust ex:grade to stu:grade) (Fig. 2–C).

SHACL Parsing. We compute the XPath for each *XSD-driven* shape based on shape identifiers and the referencing relationships among shapes by utilizing a depth-first search (DFS) algorithm. The identifier for shapes (e.g., http://example.com/PropertyShape/studentType/grade) generated by XSD2SHACL is the concatenation of the names of element declarations (e.g., grade), attribute declarations (e.g., id), and global complex type definitions (e.g., studentType) within the XSD. It encapsulates the path from the global declaration or definition (e.g., studentType) to the corresponding XSD component (e.g., grade). Nevertheless, the identifier cannot be swiftly extracted as an XPath due to the potential truncation of the XPath (e.g., /student/studentType and /studentType/grade) in the identifier. Thus, the entire shapes graph needs to be traversed to compute the complete XPath (e.g., student/grade).

We convert the shapes into a reversed directional acyclic graph of identifiers and employ a DFS algorithm to traverse it, then compute the complete XPath for each shape by considering all relevant shapes. We initiate the process by reversing the relationships of identifiers referenced through sh:property and sh:node to construct a graph. Shapes that reference other shapes but are not referenced by any



```
<student id="001">
  <grade> 20 </grade>
</student>
```

```
<xsd:element name="student" type="studentType"/>

<xsd:complexType name="studentType">
  <xsd:all>
    <xsd:element name="grade" type="xsd:integer>
  </xsd:all>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>
```

```
StudentMapping:
 sources:
  - ["student.xml~ql:XPath" , "/student"]
 s: "ex:@id"
 po:
  - [rdf:type, stu:Student ]
  - [stu:grade, "/grade"; xsd:int ]
```

```
{"StudentMapping": {
   "subject map": {
      "class": [stu:Student]
      "path": ["/student/@id"]  }
   "predicate-object map":[
     {"property": [stu:grade],
      "path": [/student/grade] }
     ]  }
```

```
ex:NodeShape/student a sh:NodeShape ;
   sh:nodeKind sh:IRI ;  sh:name "student" ;
   sh:node ex:NodeShape/studentType> ;
   sh:targetClass ex:student .

ex:NodeShape/studentType a sh:NodeShape ;
   sh:name "studentType" ;
   sh:property ex:PropertyShape/studentType/grade,
      ex:PropertyShape/studentType/@id .

ex:PropertyShape/studentType/grade a sh:PropertyShape;
   sh:datatype xsd:integer ;  sh:name "grade" ;
   sh:minCount 1 ;  sh:maxCount 1 ;
   sh:path ex:grade .

ex:PropertyShape/studentType/@id a sh:PropertyShape ;
   sh:datatype xsd:string ;  sh:name "id" ;
   sh:path ex:id .
```

```
{
"ex:NodeShape/student":
          "/student",
 "ex:NodeShape/studentType":
          "/student/studentType",
"ex:PropertyShape/studentType/grade":
          "/student/grade",
 "ex:PropertyShape/studentType/@id":
          "/student/@id"
}
```

```
ex:NodeShape/student a sh:NodeShape ;
   sh:nodeKind sh:IRI ;  sh:name "student" ;
   sh:node ex:NodeShape/studentType ;
   sh:targetClass stu:Student .

ex:NodeShape/studentType a sh:NodeShape ;
   sh:name "studentType" ;
   sh:property ex:PropertyShape/studentType/grade.

ex:PropertyShape/studentType/grade a sh:PropertyShape;
   sh:datatype xsd:integer ;  sh:name "grade" ;
   sh:minCount 1 ;  sh:maxCount 1 ;
   sh:path stu:grade.
```

Fig. 2: An example of **post-adjustment**. RML in YARRRML serialisation.

shape are transformed into leaf nodes (e.g., http://example.com/NodeShape/student). Then we traverse the graph using DFS to calculate the paths leading to all potential leaf nodes commencing from each node. We can distinguish the segments within identifiers that are not elements or attributes names (e.g., studentType), as complex type definitions do not require the target declarations creation in XSD2SHACL (e.g., http://example.com/NodeShape/student/studentType). Ultimately, we obtain an XPath for each shape by concatenating the path fragments found within each path list (e.g., /student and /grade), while eliminating any fragments corresponding to complex type definitions (e.g., studentType). An example outcome of SHACL parsing can be found in Fig. 2.

RML Parsing. We parse the RML mappings to extract the XPath (e.g., student/grade) corresponding to the associated classes or properties (e.g., stu:grade) as Fig. 2 shows. For each triples map (e.g., :StudentMapping), we identify all relevant classes (e.g., stu:Student) and associate them with the XPath extracted from a template (e.g., id from http://example.com/@id) or reference in the subject map and concatenate it with the iterator (e.g., /student). Similarly, for each predicate-object map, we correlate the property (e.g., stu:grade) defined by the predicate map with the XPath (e.g., /student/grade) extracted also from the object map, as described previously. Simultaneously, in cases of multiple components templates (e.g., http://example.com/{@id}_{name}) where multiple XPaths are implicated and multiple shapes are involved, we manage them as a list (e.g., [/student/@id, /student/name]) for future adjustment.

Adjustment. We adjust the classes, properties, and constraints within the preliminary shapes to align with the target RDF graph based on parsing outcomes from both RML and SHACL, and showcase the example in Fig. 2. For each class and property (e.g., stu:grade) associated with the XPath (e.g., /student/grade) within the RML parsing results, we look for the corresponding shape identifier (e.g., http://example.com/PropertyShape/studentType/grade) that shares

Table 1: Focus node and property value extraction for equivalences identification.

Extraction	**Resultant mapping**
?S a sh:NodeShape . ?S sh:targetClass | sh:targetNode | sh:targetSubjectsOf | sh:targetObjectsOf ?node.	focusNode[S]=node
?S sh:node* ?S2.	focusNode[S2]+=focusNode[S]
?S a sh:PropertyShape . ?S sh:targetClass | sh:targetNode | sh:targetSubjectsOf | sh:targetObjectsOf ?fn. ?S sh:path ?node.	propertyValue[S]=(?fn,?node)
if focusNode[S]: ?S sh:property ?PS. ?PS sh:path ?node.	propertyValue[PS]=(focusNode[S],?node)

the same XPath in the SHACL parsing results. If found, we update the target declaration (i.e. sh:targetClass) or property path (i.e. sh:path) with the class or property (e.g., stu:grade) defined in RML. Ultimately, we eliminate shapes that remain unadjusted and are not referenced by the adjusted shapes.

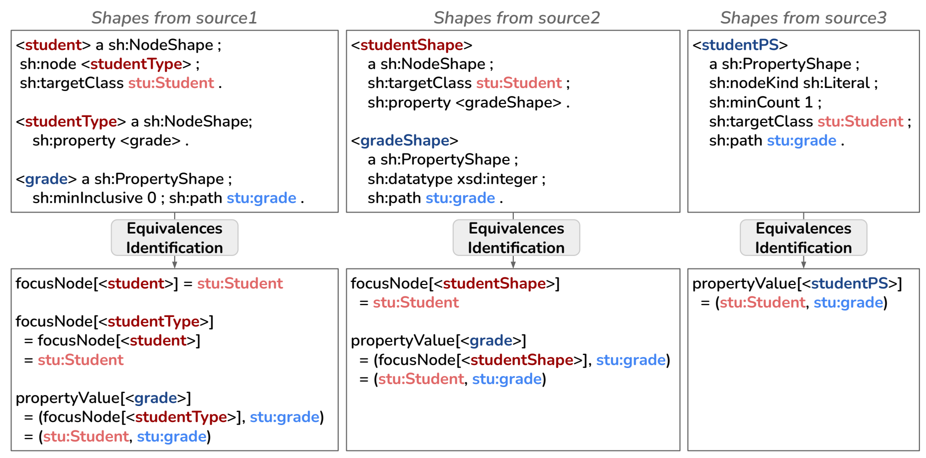
In addition to aligning classes and properties, we specifically create a new shape for templates with multiple references to the raw data schema (e.g., http://example.com/{id}_{name}). As that template involves multiple shapes derived from the raw data schema, one for each XPath expression in the template, the constraints originally present in individual shapes may no longer be valid. For instance if, on the one hand, we have string-based constraints (e.g., sh:pattern) and, on the other hand, the sh:datatype is an integer, not both constraints may be retained. After parsing the RML mappings, the XPath list is retrieved.

Moreover, in the presence of cardinality constraints (e.g., sh:minCount 2 and sh:minCount 3) across all pertinent shapes, these constraints are multiplied to calculate the cardinality constraint (e.g., sh:minCount 6) for the merged shape. Similarly, if length constraints are present in all relevant shapes (e.g., sh:minLength 3 for id and sh:minLength 1 for name in the template above), the cumulative sum (e.g., sh:minLength 24), considering the length of the string outside the component in the template (e.g., length of 20 for http://example.com/_), is added.

4.2 Equivalences Identification

To determine the constraints within shapes that should be integrated, we identify the equivalent shapes, i.e. the shapes designated for validating either the same focus node or the identical property value reached via the same focus node.

We traverse the shapes from diverse sources that are intended for integration, sequentially evaluating the following four scenarios, as delineated in Table 1, to ascertain equivalency: (i) the present focus nodes of the node shape that defines the target declaration; (ii) the cumulative focus nodes of shapes referenced through sh:node by other shapes that possess focus nodes; (iii) the directly defined property path with focus nodes; and (v) the property path of the property shape referenced by the node shape with focus nodes through sh:property. Utilizing these outcomes, shapes sharing identical focus nodes or property values, derived from disparate sources, are recognized as equivalent shapes.


Shapes from source1
Shapes from source2
Shapes from source3
Equivalences Identification
Equivalences Identification
Equivalences Identification

```
<student> a sh:NodeShape ;
 sh:node <studentType> ;
 sh:targetClass stu:Student .

<studentType> a sh:NodeShape;
   sh:property <grade> .

<grade> a sh:PropertyShape ;
   sh:minInclusive 0 ; sh:path stu:grade .
```

```
<studentShape>
   a sh:NodeShape ;
   sh:targetClass stu:Student ;
   sh:property <gradeShape> .

<gradeShape>
   a sh:PropertyShape ;
   sh:datatype xsd:integer ;
   sh:path stu:grade .
```

```
<studentPS>
   a sh:PropertyShape ;
   sh:nodeKind sh:Literal ;
   sh:minCount 1 ;
   sh:targetClass stu:Student ;
   sh:path stu:grade .
```

```
focusNode[<student>] = stu:Student

focusNode[<studentType>]
  = focusNode[<student>]
  = stu:Student

propertyValue[<grade>]
  = (focusNode[<studentType>], stu:grade)
  = (stu:Student, stu:grade)
```

```
focusNode[<studentShape>]
  = stu:Student

propertyValue[<grade>]
  = (focusNode[<studentShape>], stu:grade)
  = (stu:Student, stu:grade)
```

```
propertyValue[<studentPS>]
  = (stu:Student, stu:grade)
```

Fig. 3: An example process of the **equivalences identification** module. Identifiers of the same color refer to equivalent shapes.

Figure 3 presents an example of equivalences identification, emphasizing the necessity of considering four scenarios. The first scenario's equivalences calculation facilitates the determination of the focus nodes directly defined by node shapes (e.g., <student> sh:targetClass stu:Student). The second scenario's equivalence calculation prevents the oversight of focus nodes indirectly defined through sh:node reference relationships (e.g., <student> sh:node <studentType>). The third scenario assists in identifying the focus node and property value pair that is directly defined within the same property shape (e.g., <studentPS> sh:targetClass stu:Student, sh:path stu:grade). The fourth scenario aids in determining the property value reached through the sh:property reference relationship from the focus node (e.g., <studentShape> sh:property <gradeShape>).

4.3 Integration and Inconsistencies Resolution

To provide a variety of integration possibilities, we devised three approaches: SCOOP-All, SCOOP-Prior, and SCOOP-Prior-R. *All* aims to integrate all constraints from all equivalent shapes, *Prior* aims to integrate constraints from lower-priority shapes that are not inconsistent with higher-priority equivalent shapes based on defined source priority, and *R* is designed to filter out redundant shapes from the ontology based on higher-priority shapes. We also introduce the inconsistencies resolutions for *All* and *Prior* to harmonize all constraints within equivalent shapes to be integrated, mitigating inconsistencies in the final shape.

SCOOP and Resolution. For all algorithms, we initialize an empty graph S to store the final unified shapes. We consider the current shape s as the shape to be added and the current constraint c as the constraint to be added. The constraints defined for shapes equivalent to s in S are denoted as $C(s)$. For SCOOP-All, we sequentially add s translated from different sources. If an equivalent shape does not exist in S, we directly add s to S. Otherwise, we

Table 2: The inconsistent combination of constraints slated for addition and the corresponding constraints within the presently integrated shape.

Constraints to be added	**Inconsistent constraints**
sh:datatype	sh:nodeKind *isnot* sh:Literal
sh:nodeKind *is* (sh:IRIOrLiteral *or* sh:IRI *or* sh:BlankNodeOrLiteral *or* sh:BlankNode *or* sh:BlankNodeOrIRI)	sh:datatype *or* sh:languageIn *or* sh:minExclusive *or* sh:minInclusive *or* sh:maxExclusive *or* sh:maxInclusive
sh:minCount	$>$ sh:maxCount
sh:maxCount	$<$ sh:minCount
sh:minExclusive	sh:nodeKind *isnot* sh:Literal *or* $\geq$ sh:maxExclusive *or* $\geq$ sh:maxInclusive
sh:minInclusive	sh:nodeKind *isnot* sh:Literal *or* $\geq$ sh:maxExclusive *or* $>$ sh:maxInclusive
sh:maxExclusive	sh:nodeKind *isnot* sh:Literal *or* $\leq$ sh:minExclusive *or* $\leq$ sh:minInclusive
sh:maxInclusive	sh:nodeKind *isnot* sh:Literal *or* $\leq$ sh:minExclusive *or* $<$ sh:minInclusive
sh:minLength	sh:nodeKind *isnot* sh:Literal *or* $>$ sh:maxLength
sh:maxLength	sh:nodeKind *isnot* sh:Literal *or* $<$ sh:minLength

perform constraint checks on $C(s)$ and c (explained in detail below). In cases where no inconsistencies are found, we add c directly to $C(s)$. Conversely, we reconstruct and integrate inconsistent constraints using sh:or. For SCOOP-Prior, we traverse and add s into S based on the user-defined priorities of different sources. Similarly, if an equivalent shape does not exist in S, we directly add s to S, otherwise, we conduct an inconsistencies check. However, in cases of finding an inconsistency, we refrain from adding c. Unlike SCOOP-Prior, SCOOP-Prior-R refrains from adding s that is translated from ontology to S if there is no equivalent shape in S.

Inconsistencies Check. We perform two types of constraint checks: (i) internal checks based on the SHACL specification [27], determining whether adding c violates the constraint's cardinality restriction (e.g., a shape cannot have two sh:minLength), and (ii) external checks based on Table 2, verifying if there are inconsistencies between $C(s)$ and c (e.g., $C(s)$ contains sh:nodeKind sh:IRI while c defines sh:datatype xsd:string). Specifically, we categorized the core constraints into those constraints permitting at most one appearance in a shape and those allowing multiple occurrences. We consider it an inconsistency if there is a constraint in $C(s)$ same as c that allows only one occurrence. All cases are outlined in Table 2.

4.4 Implementation

SCOOP integrates RML2SHACL [10], Astrea [5], and XSD2SHACL [13] to extract shapes from the mappings, ontology, and XSD respectively, implements the post-adjustment, equivalences identification, and integration and inconsistencies resolution modules in the workflow (Fig. 1) as outlined in Algorithm 1, and provides configuration options with SCOOP-All, SCOOP-Prior, and SCOOP-Prior-

Algorithm 1: SCOOP implementation.

```
Function SCOOP($rml\_files = \emptyset$, $owl\_files = \emptyset$, $xsd\_files = \emptyset$,
  $priority = ["rml", "owl", "xsd"]$, $mode = ["All", "Prior", "Prior\_R"]$):
    $shapes\_inorder$, $S \leftarrow \emptyset$, Graph()
    for $source$ in $priority$ do
        if $source$ is $rml$ and $rml\_files$ then
            $shapes \leftarrow$ RML2SHACL($rml\_files$)
        else if $source$ is $owl$ and $owl\_files$ then
            $shapes \leftarrow$ OWL2SHACL($owl\_files$)
        else if $source$ is $xsd$ and $xsd\_files$ then
            $xsd\_shapes \leftarrow$ XSD2SHACL($xsd\_files$)
            if $rml\_files$ then
                $xsd\_shapes \leftarrow$ Post_adjustment($shapes$, $rml\_files$)
        $shapes\_inorder$.add($shapes$)
    $S \leftarrow shapes\_inorder[0]$
    for $shape\_next$ in $shapes\_inorder[1:]$ do
        $parseResults\_current \leftarrow$ Equivalences_identification($S$)
        $parseResults\_next \leftarrow$ Equivalences_identification($shape\_next$)
        for $parseResult$ in $parseResults\_next$ do
            if $parseResult$ can be found in $parseResults\_current$ then
                $S \leftarrow$ Inconsistencies_resolution_integration($S$, $shape\_next$,
                  $parseResult$, $parseResults\_current$) // Check for inconsistencies
                  that may caused by constraints to be added and resolve them
            else if $mode$ is not $Prior\_R$ then
                $S \leftarrow$ addConstraints($shape\_next$, $parseResult$) // Direct add the
                  corresponding shape with constraints since there is no equivalent
                  shape that may raise inconsistencies found in the current shape
    return $S$
```

R. SCOOP accepts files from diverse sources as input, with default user-defined priority that accords precedence to mappings over ontology and over XSD, ultimately yielding a unified shapes graph. Besides, the SCOOP framework can be expanded to incorporate shapes from supplementary sources, such as CSVW and JSON Schema. The code and usage instructions for SCOOP are available online on the GitHub repository: `https://github.com/dtai-kg/SCOOP`.

5 Evaluation

We performed a comprehensive analysis, encompassing both performance and coverage aspects, to evaluate the effectiveness of SCOOP in addressing real-world use cases and compared it with current state-of-the-art systems.

Systems. We selected a state-of-the-art system from each of the *RDF graphs*, *ontologies*, *mappings*, and *raw data schemas*-based systems for extracting SHACL shapes to serve as a point of comparison. For the *RDF graphs*-based system, we considered the available systems, including SHACLGEN [2], SheXer [1], and QSE [31], ultimately opting for QSE. This selection was influenced by QSE's broader coverage of constraints compared to other systems [32]. For the *ontologies*-based system, our considerations encompass Astrea [5] and the OWL2SHACL conversion rules [21]. We finally selected Astrea due to the availability of detailed information on its implementation and performance [7], which is absent for OWL2SHACL. For the systems based on *mapping rules* and *raw data schemas*, we opted for RML2SHACL [10] and XSD2SHACL [13].

Datasets. To demonstrate our framework's potential, we leverage the real-world use case RINF [36], the railway infrastructure register published by each country, containing the infrastructure parameters applied to the railway system. RINF includes raw XML data from 30 countries, 28 RML files[2], the ERA ontology version 3.0.0 [16], an OWL ontology, and the RINF XSD Schema version 1.5 [37]. We arrange the XML data from 30 countries in ascending order of size and progressively construct RDF graphs using the RML mapping rules and the RMLMapper [25]. The minimum RDF graph (0.003 GB) comprises solely the smallest XML dataset, while the largest RDF graph (8.52 GB) incorporates data from all countries. The constructed RDF graphs serve as input for QSE.

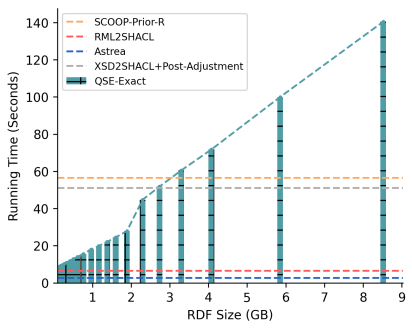
Experimental Setup. We conduct all experiments on a Rocky 8.8 Linux based system with Intel(R) Xeon(R) Gold 6140 CPUs running at 2.30 GHz, configured with 128 GB RAM and 32 cores. We experimented with SCOOP v1.0.0 in its different configurations (i.e. SCOOP-All, SCOOP-Prior, and SCOOP-Prior-R) under different input combinations. We executed the RML2SHACL v1.0.0, Astrea v1.2.1, and XSD2SHACL v1.0.0 systems as per the instructions outlined in their repositories. Regarding QSE (*pvldb* release), we ran QSE-Exact with confidence $\geq 25\%$ and support $\geq 100\%$, and QSE-Approx with confidence $\geq 25\%$, support $\geq 100\%$, sampling percentage of 50%, and reservoir size of 5000.

Performance Analysis. We evaluated the running time and maximum memory usage of different systems (average of 5 runs) in the RINF use case across diverse data sizes. Fig. 4 illustrates the running time and maximum memory usage of different systems, with both figures including only one QSE-Exact and one SCOOP-Prior-R due to negligible differences between the various approaches. Fig. 4b shows QSE and SCOOP exclusively, as the results for other systems align closely with those of SCOOP.

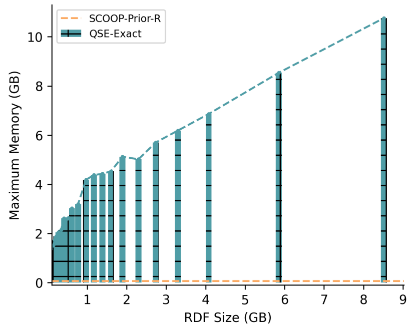
Our experiments demonstrate that the memory usage of the QSE increases approximately linearly with the size of the RDF graph, and the maximum memory usage of QSE is higher than SCOOP in all experiments. The running time of QSE is higher than all non-RDF-based systems without post-adjustment when the RDF graph's size exceeds 0.10GB; is higher than XSD2SHACL with post-adjustment when the graph size exceeds 2.73GB; and is higher than SCOOP when the RDF graph's size exceeds 3.30GB. The running time of SCOOP is primarily influenced by the post-adjustment phase, while the triggering and integration of shapes incur relatively less time. SCOOP's memory usage is independent of the RDF graph's size. SCOOP uses lower memory usage than QSE throughout all experiments, and surpasses QSE in terms of running time when the RDF graph's size exceeds 3.30 GB, ensuring scalability for larger RDF graphs.

Coverage Analysis. We conducted a statistical analysis of the extracted shapes concerning targeting sufficient classes and properties and generating rich constraints. We extracted the involved 21 classes and 223 properties of the largest

[2] The mapping rules will be publicly available soon by ERA (`era.europa.eu/`).



(a) Running time.



(b) Maximum memory usage.

Fig. 4: (a) Running time and (b) Maximum memory usage comparison across various systems on the RINF use case.

RDF graph (8.52 GB), which includes all raw data as the ground truth. We also chose the shapes produced by QSE on this RDF graph for comparison.

We compared the classes defined by the sh:targetClass and the properties defined by the sh:path referenced by shapes containing target declarations in the extracted shapes with the ground truth to calculate precision (P), recall (R), and F1 score. We also assessed the richness of constraints within shapes. Following the SHACL specification [28], we categorized SHACL core constraints into eight categories: Value Type (VT), Value Range (VR), String-based (SR), Property Pair (PP), Logical (LG), Shape-based (SA), Other (OT) which also includes sh:name and sh:description, and analyze the coverage for each category. Table 3 show that SCOOP-Prior-R performs comparably to XSD2SHACL with our post-adjustment module, and surpasses all other systems in defining sufficient classes and properties. SCOOP-All generates the richest constraints, and all SCOOP systems surpass QSE by generating more than twice the types of constraints.

Coverage Analysis in Classes. Examining the outcomes for the targeted classes, QSE and systems involving RML demonstrate the capability of perfectly targeting classes. The exceptional performance of QSE in class extraction is foreseeable, given its input is the target RDF graph. Among the non-RDF-systems, RML2SHACL benefits from RML, aligning perfectly with the target RDF graph's classes. Our implemented post-adjustment helps XSD2SHACL to capture information from RML, thereby achieving full scores across all metrics. However, Astrea exhibits lower precision, a common issue in systems extracting shapes from ontologies, because ontologies cover the entire domain but the target RDF graph may only involve a subset, resulting in many unnecessary shapes. In the case of the SCOOP-All and SCOOP-Prior, the inclusion of the entire ontology negatively impacts the precision where SCOOP-Prior-R is specifically designed to mitigate the issue of redundant shapes generated by ontologies. The results demonstrate that SCOOP-Prior-R achieves full scores across all metrics for all input combinations, indicating its stronger robustness.

Table 3: The precision (P), recall (R), and F1 score on the extracted classes and properties, and the coverage of constraints: Value Type (VT), Cardinality Constraint (CD), Value Range (VR), String-based (SR), Property Pair (PP), Logical (LG), Shape-based (SA), Other (OT). *PA* refers to the post-adjustment.

Systems		Classes			Properties			Core Constraints								
		P	R	F1	P	R	F1	VT	CD	VR	SR	PP	LG	SA	OT	Total
	QSE-Exact	**1.0**	**1.0**	**1.0**	**1.0**	0.44	0.61	2	1	0	0	0	0	1	**2**	7
	QSE-Approx	**1.0**	**1.0**	**1.0**	**1.0**	0.44	0.61	2	1	0	0	0	0	1	**2**	7
	RML2SHACL	**1.0**	**1.0**	**1.0**	**1.0**	0.26	0.41	2	0	0	1	0	0	2	0	5
	Astrea	0.45	**1.0**	0.62	0.44	0.94	0.60	**3**	1	0	1	**1**	**2**	4	**2**	14
	XSD2SHACL+*PA*	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.99**	2	**2**	0	2	0	0	2	1	9
SCOOP All	*rml+owl*	0.45	**1.0**	0.62	0.45	0.98	0.62	**3**	1	0	1	**1**	**2**	**5**	**2**	15
	rml+xsd	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.99**	2	**2**	0	**3**	0	1	2	1	11
	rml+owl+xsd	0.45	**1.0**	0.62	0.45	**1.0**	0.62	**3**	**2**	0	**3**	**1**	**2**	**5**	**2**	**18**
SCOOP Prior	*rml+owl*	0.45	**1.0**	0.62	0.45	0.98	0.62	**3**	1	0	1	**1**	1	**5**	**2**	14
	rml+xsd	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.99**	2	**2**	0	**3**	0	0	2	1	10
	rml+owl+xsd	0.45	**1.0**	0.62	0.45	**1.0**	0.62	**3**	**2**	0	**3**	**1**	1	**5**	**2**	17
SCOOP Prior-R	*rml+owl*	**1.0**	**1.0**	**1.0**	**1.0**	0.26	0.41	**3**	1	0	1	0	0	**5**	**2**	12
	rml+xsd	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.99**	2	**2**	0	**3**	0	0	2	1	10
	rml+owl+xsd	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.99**	**3**	**2**	0	**3**	0	0	**5**	**2**	15

Coverage Analysis in Properties. In terms of targeted properties, SCOOP exhibits superior overall performance compared to other systems. The low recall scores of QSE indicate that they omit many properties during the shape extraction process. We attribute this to their novel extraction method based on support and confidence score, which, in an attempt to filter out spurious shapes, also filter out some properties present in the RDF graph. The low recall values observed in RML2SHACL can be attributed to instances where certain property shapes lack references from target declarations. This is attributed to implicit class definitions in our RML use case, where multiple triples maps are involved, with only one explicitly defining the class, while the others imply its existence using the same subject map. However, RML2SHACL fails to capture this implicit definition, resulting in the oversight of certain reference relationships. In contrast, our post-adjustment module combined with XSD2SHACL performs well and does not encounter this issue because we considered implicit class definitions when parsing RML, and the precision of 0.98 suggests that the generated RDF graph may not have produced some expected triples due to missing raw data. In addition, Astrea's lower precision score in property extraction is ascribed to the ontology's broader coverage of properties. The performance of all SCOOP approaches with RML and XSD as inputs overcomes all other systems.

Coverage Analysis in Constraints. The analysis of core constraints confirms the outstanding performance of SCOOP in terms of constraint richness. The shapes extracted by SCOOP-All encompass a total of 18 distinct constraints, slightly surpassing SCOOP-Prior, SCOOP-Prior-R, and Astrea, marginally higher than XSD2SHACL, and markedly superior to both QSE and RML2SHACL. Furthermore, with regard to the SCOOP-All, we conducted a fine-grained analysis of the contributions of different sources to the final constraints. Among the 18 final constraints, 5 originate from RML, 14 from the ontology, and 9 from XSD. In particular, the ontology covers property pair constraints (e.g., sh:equals) and logical constraints (e.g., sh:not) not covered by other sources.

Practical Implications on Validation. We assessed the effectiveness of SCOOP to extract valid SHACL shapes by validating an RDF graph using those shapes. We used the pySHACL validator [38] to validate the smallest RDF graph (0.003 GB, 1620 triples) constructed from RINF. Due to violations of SHACL syntax in shapes generated by QSE, e.g., multiple occurrences of sh:in, and Astrea, e.g., missing sh:path, we could not directly use their shapes for validation.

The validation results indicated that the target RDF graph conformed to all shapes generated by RML2SHACL but violated shapes generated by other systems. Specifically, the shapes extracted by XSD2SHACL revealed 738 missing triples and 540 error triples (e.g., sh:maxLength, sh:datatype, and sh:nodeKind). We analyzed the additional violations found by XSD2SHACL, attributing them to lingering issues from mismatched versions of XSD and RML. SCOOP-Prior and SCOOP-R, incorporating the mapping rules, ontology, and XSD, discovered 738 missing triples and 972 error triples (involving sh:class and sh:maxLength). We attribute this to the introduction of constraints from the ontology, distinguishing it from XSD2SHACL. SCOOP-All, including all sources, also identified 738 missing triples but labeled 864 error triples (involving only sh:class), slightly fewer than SCOOP-Prior. This was reasonable as SCOOP-All's strategy of adding sh:or introduces more lenient constraints to the RDF graph. In summary, this experiment demonstrates that our SCOOP framework strictly adheres to SHACL syntax, and provides users with effective RDF validation results.

6 Conclusion

In this paper, we propose SCOOP, an open-source framework designed to integrate existing shape extraction systems targeting diverse sources into a unified and comprehensive shapes graph. SCOOP consists of three modules and devises three integration approaches to provide diverse choices. The implementation of SCOOP includes the incorporation of RML2SHACL, Astrea, and XSD2SHACL. SCOOP exhibits significantly lower memory usage compared to RDF-based systems, performs faster when the RDF graph exceeds 3.30 GB in size, and outperforms state-of-the-art systems in extracting effective node and property shapes and generating richer constraints.

SCOOP is an extensible framework that goes beyond current sources and systems, including ontologies, mapping rules, and raw data schemas. The methodology is transferable to any SHACL shapes extraction work, therefore, seamless integration with other systems can be easily achieved. In the future, we will extend SCOOP to a broader range of sources, e.g., CSVW, JSONSchema, etc.

SCOOP enables users to extract SHACL shapes effortlessly, without even having to construct the RDF graphs using hardware with limited resources. It is expected that this framework will have a high impact on the Semantic Web community as most RDF graphs nowadays are constructed from various data.

Acknowledgement

Xuemin Duan and Anastasia Dimou are partially supported by Flanders Make, the research centre for the manufacturing industry, and the Flanders innovation and entrepreneurship (VLAIO) through the KG3D project. David Chaves-Fraga is funded by the Galician Ministry of Education, University and Professional Training and the European Regional Development Fund (ERDF/FEDER program) through grants ED431C2018/29 and ED431G2019/04. The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

References

1. sheXer. `https://github.com/DaniFdezAlvarez/shexer`, accessed on 10.11.2023
2. Arndt, N.: SHACLGEN. `https://github.com/AKSW/shaclgen`, accessed on 20.09.2023
3. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language – Structural Specification and Functional-Style Syntax (Second Edition). Recommendation, World Wide Web Consortium (W3C) (Dec 2012), `http://www.w3.org/TR/owl2-syntax/`
4. Boneva, I., Dusart, J., Fernández Alvarez, D., Gayo, J.E.L.: Shape Designer for ShEx and SHACL Constraints. In: Proceedings of the ISWC 2019 Satellite Tracks (Poster & Demonstrations, Industry, and Outrageous Ideas). vol. 2456, pp. 269–272. CEUR (Oct 2019)
5. Cimmino, A.: Astrea. `https://github.com/oeg-upm/astrea`, accessed on 20.09.2023
6. Cimmino, A., Fernández-Izquierdo, A., García-Castro, R.: Astrea: Automatic Generation of SHACL Shapes from Ontologies. In: European Semantic Web Conference (ESWC). Springer, Springer International Publishing (2020)
7. Cimmino, A., Fernández-Izquierdo, A., García-Castro, R.: Astrea: Automatic Generation of SHACL Shapes from Ontologies. In: European Semantic Web Conference. pp. 497–513. Springer (2020)
8. Comelli, T.: JS2SHACL - JSON Schema to SHACL conversor. `https://github.com/ThiagoCComelli/JS2SHACL-JSON-Schema-to-SHACL-conversor`, accessed on 20.09.2023
9. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Working group recommendation, World Wide Web Consortium (W3C) (Sep 2012), `http://www.w3.org/TR/r2rml/`
10. Delva, T.: RML2SHACL. `https://github.com/RMLio/RML2SHACL`, accessed on 20.09.2023
11. Delva, T., Smedt, B.D., Oo, S.M., Assche, D.V., Lieber, S., Dimou, A.: RML2SHACL: RDF Generation Taking Shape. In: Proceedings of the 11th on Knowledge Capture Conference. pp. 153–160. ACM (Dec 2021). https://doi.org/10.1145/3460210.3493562
12. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: Proceedings of the 7th Workshop on Linked Data on the Web. vol. 1184. CEUR Workshop Proceedings (2014)

13. Duan, X.: XSD2SHACL. `https://doi.org/10.5281/zenodo.8318452` (2023), accessed on 20.09.2023
14. Duan, X., Chaves-Fraga, D., Dimou, A.: XSD2SHACL: Capturing RDF Constraints from XML Schema. In: Proceedings of the 12th Knowledge Capture Conference 2023. p. 214–222. K-CAP '23, Association for Computing Machinery (2023). https://doi.org/10.1145/3587259.3627565
15. Ekaputra, F.J., Llugiqi, M., Sabou, M., Ekelhart, A., Paulheim, H., Breit, A., Revenko, A., Waltersdorfer, L., Farfar, K.E., Auer, S.: Describing and Organizing Semantic Web and Machine Learning Systems in the SWeMLS-KG. In: Pesquita, C., Jimenez-Ruiz, E., McCusker, J., Faria, D., Dragoni, M., Dimou, A., Troncy, R., Hertling, S. (eds.) The Semantic Web. vol. 13870, pp. 372–389. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33455-9_22
16. European Union Agency for Railways: ERA_vocabulary. `https://data-interop.era.europa.eu/era-vocabulary/`, accessed on 20.09.2023
17. Fallside, D., Walmsley, P.: XML Schema Part 0: Primer Second Edition. Recommendation, W3C (Oct 2004), `https://www.w3.org/TR/xmlschema-0/`
18. Felin, R., Faron, C., Tettamanzi, A.G.B.: A Framework to Include and Exploit Probabilistic Information in SHACL Validation Reports. In: The Semantic Web. Springer Nature Switzerland (2023)
19. Fernández-Álvarez, D., García-González, H., Frey, J., Hellmann, S., Gayo, J.E.L.: Inference of Latent Shape Expressions Associated to DBpedia Ontology. In: Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17$^{\text{th}}$ International Semantic Web Conference (ISWC 2018). vol. 2180. CEUR Workshop Proceedings (2018)
20. Fernandez-Álvarez, D., Labra-Gayo, J.E., Gayo-Avello, D.: Automatic extraction of shapes using sheXer. Knowledge-Based Systems **238**, 107975 (Feb 2022). https://doi.org/10.1016/j.knosys.2021.107975
21. Francart, T.: OWL2SHACL. `https://github.com/sparna-git/owl2shacl`, accessed on 10.11.2023
22. Garcia-Gonzalez, H., Labra-Gayo, J.E.: XMLSchema2ShEx: Converting XML validation to RDF validation. Semantic Web **11**(2) (2020)
23. Ghiasnezhad Omran, P., Taylor, K., Rodríguez Méndez, S., Haller, A., et al.: Towards SHACL learning from knowledge graphs. In: Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020). vol. 2721, pp. 94–99. CEUR Workshop Proceedings (2020)
24. Harshvardhan J. Pandit, Declan O'Sullivan, D.L.: Using Ontology Design Patterns to Define SHACL Shapes. In: 9th Workshop on Ontology Design and Patterns (WOP2018). vol. 2195, pp. 67–71. CEUR-WS, Monterey California, USA (2018)
25. Heyvaert, P., Meester, B.D., et al.: RMLMapper-java. `https://github.com/RMLio/rmlmapper-java`, accessed on 20.09.2023
26. Iglesias-Molina, A., Van Assche, D., Arenas-Guerrero, J., De Meester, B., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., Dimou, A.: The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In: The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, Proceedings, Part II. p. 152–175. Springer-Verlag (2023). https://doi.org/10.1007/978-3-031-47243-5_9
27. Knublauch, H., Kontokostas, D.: SHACL-SHACL. `http://www.w3.org/ns/shacl-shacl#`, accessed on 01.12.2023

28. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). Recommendation, W3C (2017), `https://www.w3.org/TR/shacl/`
29. Mihindukulasooriya, N., Rashid, M.R.A., Rizzo, G., Garcia-Castro, R., Corcho, O., Torchiano, M.: RDF Shape Induction using Knowledge Base Profiling. In: Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing (2017)
30. Pandit, H.J., O'Sullivan, D., Lewis, D.: Using Ontology Design Patterns To Define SHACL Shapes. In: Proceedings of the 9th Workshop on Ontology Design and Patterns (WOP 2018) co-located with 17th International Semantic Web Conference (ISWC 2018). vol. 2195, pp. 67–71. CEUR (2018)
31. Rabbani, K.: Quality Shapes Extraction (QSE). `https://github.com/dkw-aau/qse`, accessed on 20.09.2023
32. Rabbani, K., Lissandrini, M., Hose, K.: Extraction of Validating Shapes from very large Knowledge Graphs [Extended Version]
33. Rabbani, K., Lissandrini, M., Hose, K.: SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption. In: Companion Proceedings of the Web Conference 2022. p. 260–263. WWW '22, Association for Computing Machinery (2022). https://doi.org/10.1145/3487553.3524253
34. Rabbani, K., Lissandrini, M., Hose, K.: Extraction of Validating Shapes from Very Large Knowledge Graphs. Proceedings of the VLDB Endowment **16**(5) (2023)
35. Rabbani, K., Lissandrini, M., Hose, K.: SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes. In: Proceedings of the 2023 International Conference on Management of Data, (SIGMOD-Companion '23). pp. 151–154. Association for Computing Machinery (2023). https://doi.org/10.1145/3555041.3589723
36. RINF, T.: RINF: Railway infrastructure register. `https://www.rinf-ch.ch/`, accessed on 01.12.2023
37. RINF, T.: RINF XML Schema v1.5. `https://www.era.europa.eu/domains/registers/rinf_en`, accessed on 01.12.2023
38. Sommer, A., Car, N.: pySHACL. `https://github.com/RDFLib/pySHACL` (Jan 2022). https://doi.org/10.5281/zenodo.4750840, `https://github.com/RDFLib/pySHACL`
39. Spahiu, B., Maurino, A., Palmonari, M.: 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). CEUR Workshop Proceedings, vol. 2195, pp. 52–66. CEUR (2018)
40. Thapa, R.B., Giese, M.: A Source-to-Target Constraint Rewriting for Direct Mapping. In: The Semantic Web – ISWC 2021. pp. 21–38. Springer (2021). https://doi.org/10.1007/978-3-030-88361-4_2