Semantic Web 0 (0) 1
IOS Press

Visual Notations for Viewing and Editing RDF Constraints with UnSHACLed

Sven Lieber [*], Ben De Meester, Pieter Heyvaert, Femke Brückmann, Ruben Wambacq, Erik Mannens, Ruben Verborgh, and Anastasia Dimou
IDLab, Department of Electronics and Information Systems, Ghent University–imec, Belgium

Abstract. The quality of knowledge graphs can be assessed by a validation against specified constraints, typically use-case specific and modeled by human users in a manual fashion. Visualizations can improve the modeling process as they are specifically designed for human information processing, possibly leading to more accurate constraints, and in turn higher quality knowledge graphs. However, it is currently unknown how such visualizations support users when *viewing* RDF constraints as no scientific evidence for the visualizations' effectiveness is provided. Furthermore, some of the existing tools are likely suboptimal, as they lack support for *edit operations* or common constraints types. To establish a baseline, we have defined visual notations to view and edit RDF constraints, and implemented them in *UnSHACLed*, a tool that is independent of a concrete RDF constraint language. In this paper, we (i) present two visual notations that support all SHACL core constraints, built upon the commonly used visualizations VOWL and UML, (ii) analyze both notations based on cognitive effective design principles, (iii) perform a comparative user study between both visual notations, and (iv) present our open source tool UnSHACLed incorporating our efforts. Users were presented RDF constraints in both visual notations and had to answer questions about it. Although no statistical significant difference in mean error rates was observed, a majority of participants made less errors with *ShapeVOWL* and all preferred *ShapeVOWL* in a self-assessment to answer RDF constraint-related questions. Study participants argued that the increased visual features of *ShapeVOWL* made it easier to spot constraints, but a list of constraints – as in *ShapeUML* – is easier to read. However, also that more deviations from the strict UML specification and introduction of more visual features can improve *ShapeUML*. From these findings, we conclude that *ShapeVOWL* has the potential for more user acceptance, but also that the clear and efficient text encoding of *ShapeUML* can be improved with visual features. A one-size-fits-all approach to RDF constraint visualization and editing will be insufficient. Therefore, to support different audiences and use cases, user interfaces of RDF constraint editors need to support different visual notations. In the future, we plan to incorporate different editing approaches and non-linear workflows into *UnSHACLed* to improve its *editing* capabilities. Further research can built upon our findings and evaluate a *ShapeUML* variant with more visual features or investigate a mapping from both visual notations to ShEx constraints.

Keywords: Visual Notation, Data Shapes, Constraints, SHACL, UML, VOWL

1. Introduction

Data interoperability is one of the biggest challenges of the current era and the Resource Description Framework (RDF) offers a solution as it is compositional: RDF graphs from different sources can be merged automatically which facilitates the integration of heterogeneous data [1]. However, advantages such as RDF's flexibility also result in challenges such as the production/consumption dilemma [1] in which the structure of data needs to be de described such that producers and consumers can validate transmitted data for reasons such as security or performance [1]. In 2017, the W3C *RDF Data Shapes Working Group* published a recommendation to define structural constraints of RDF data [2] which can address such needs.

Quality is defined as "fitness for use" [3] implying that constraints for validation are use-case specific; human users usually define these constraints in a manual fashion and need support. Users can use any text editor to create such constraints, but need to be familiar with the textual syntax of the underlying data shape language. User evaluations of visualizations for different Linked Data concepts, such as ontology modeling [4]

[*]Corresponding author. E-mail: sven.lieber@ugent.be.

or Linked Data generation [5], suggest that such visualizations support users to perform respective tasks more intuitively. However, the degree of actual support offered by existing visualizations for RDF constraints is currently unknown, given the lack of scientific evidence for their effectiveness. Furthermore, some of the existing tools are likely suboptimal, as they lack support for edit operations or common constraints types.

Clearly specified visualizations – already used for some Semantic Web concepts [4–7] – provide a design rationale and can be designed with the human information processing system in mind [8], but are not yet taken into account for RDF constraints which makes the effectiveness of existing tools questionable. A visual notation [8] is defined as a set of graphical symbols, a set of compositional rules, as well as the definitions and meaning of each symbol, and provides an explicit design rationale. UnSHACLed [9], a tool built on top of SHACL [2], lists features for a visual data shape editor. However, important details regarding the used visual notation are not provided, for instance, the meaning of arrows or the selection of colors are not clearly specified. Similarly, RDFShape which uses "UML-like class diagrams" [10] to statically visualize ShEx [11] constraints does not provide a clear specification of its visual notation and neither do other recently developed tools[1] [2].

Existing tools only provide limited or no *editing capabilities*, if editing capabilities are provided they are not always in line with real-life constraint use. The first version of UnSHACLed supports constraints editing. However, it does not support all constraint types, for instance, logical constraints are not yet visualized. RDFShape does not support constraints *editing* at all as it only visualizes constraints, thus users need to use and understand the underlying textual syntax. Similar to the initial version of UnSHACLed, the implementation of RDFShape does not yet support logical relationships such as (exclusive) disjunction; recent statistics show that *disjunction constraints* are broadly used [12] and thus users probably have the need to *create and edit* such constraints.

1.1. Research question and approach

The aforementioned motivate our high-level research question: *How can we support users familiar with Linked Data in viewing and editing RDF constraints?* To address this research question, we investigated *visual notations* supporting users when *viewing* RDF constraints and present a new version of our tool *UnSHACLed* that implements visual notations and allows users to *create and edit* RDF constraints.

A few visual notations already exist, but are not formally defined or do not cover all SHACL core constraints which also prevents a fair comparison. Thus, we defined two visual notations to represent all SHACL core constraints and related concepts by reusing existing notations. Different candidates to reuse exist, i.e. commonly used visual notations already familiar to users. Both the Unified Modeling Language (UML) [13] and the Visual Notation for OWL Ontologies (VOWL) [4] can be considered for a visual notation for RDF constraints as they are commonly used for RDF constraints or related Semantic Web concepts [4–6, 9, 10, 14, 15].

1.2. Hypothesis

We defined the two visual notations *ShapeUML* and *ShapeVOWL* both representing all SHACL core constraints and related concepts. Since *VOWL*, the underlying notation of *ShapeVOWL* aims to be intuitive and comprehensible [4] and visualizes the tangible graph structure of RDF, we investigate in this paper the following hypothesis: **users familiar with Linked Data can answer questions about visually represented RDF constraints more effective with ShapeVOWL than with ShapeUML**.

1.3. Contributions

We compare the notations with respect to design principles for visual notations [8] and evaluate them in a comparative user study. We implemented both visual notations in *UnSHACLed* to allow *creating and editing* constraints in a constraint language independent way. Users can switch between visual notations and use the created RDF constraints to validate input data from within the same editor.

Our contributions in this paper are:

1. introduction of two alternative visual notations: *ShapeUML* and *ShapeVOWL*;
2. analysis of both visual notations with respect to cognitive effective design principles;

[1] OntoPad: https://web.archive.org/web/20201104091304/https://github.com/AKSW/OntoPad/

[2] shaclEditor: https://web.archive.org/web/20201104091927/https://github.com/firmao/shaclEditor

3. comparative evaluation between ShapeVOWL and ShapeUML with a user study; and
4. presentation of our open source UnSHACLed editor implementing both visual notations.

The comparative analysis based on cognitive effective design principles [8] reveals that *ShapeVOWL* adheres to more principles, thus in theory is more cognitive effective. An additional comparative user study shows that there is no significant mean error difference when answering questions about RDF constraints with both notations, however, also that in a self-assessment users prefer *ShapeVOWL*. We implemented both visual notations in our tool *UnSHACLed* to also allow editing of RDF constraints in a visual fashion.

The remainder of the paper is structured as follows. We provide background knowledge on data shape languages and visual notations in Section 2 and present two visual notations in Section 3. In Section 4 we compare both presented visual notations based on design principles for cognitive effective visualizations. In Section 5 we present our visual editor UnSHACLed. In Section 6 we present the comparative user evaluation and its results. We discuss and conclude in Section 7.

2. State of the Art

In this section, we discuss (i) existing RDF constraint languages (ii) the use of different constraint types suggesting visualizations for manual creation, (iii) existing RDF constraint visualization tools, and (iv) closely related Semantic Web visualizations providing possible visualizations to extend.

2.1. RDF constraint languages

Several RDF constraint languages were proposed in the past, we describe how they are related. In this work we consider the Shapes Constraint Language (SHACL) because it (i) is a W3C recommendation, (ii) clearly defines constraint types in its core specification, and (iii) has a significant intersection with the Shape Expressions Language (ShEx) [1], a widely used RDF constraint language.

SPARQL Inference Notation (SPIN) [16] was the earliest W3C member submission (2011). A syntax and a vocabulary were defined to describe constraints and inference rules based on SPARQL.

A few years later in 2014 another two W3C member submission were submitted: the **Resource Shape (ReSh)** [17, 18] which defines a high-level RDF vocabulary to specify the shape of RDF resources and the grammar-based **Shape Expressions Language (ShEx)** [19]. ShEx was inspired by ReSh yet provides more expressivity [11].

The **Shapes Constraint Language (SHACL)** [2] became a W3C recommendation in 2017 and is seen as the legitimate successor of SPIN [20]. SHACL is a constraint language for describing and validating RDF graphs. It defines a RDF vocabulary to define constraints and a specified validation process to validate RDF data based on described constraints: data graph nodes are validated with data shape graph constraints and a validation report in RDF following the SHACL vocabulary is generated. Furthermore, SHACL provides 31 core constraint types and other concepts related to validation both defined using the aforementioned vocabulary. These other concepts comprise (i) *a targeting mechanism* to assign data graph nodes to data shape graph constraints, (ii) *property paths* to further specify on which reachable node properties constraints apply, (iii) *severity* of data shapes as annotation to indicate the severity of a constraint violation in the validation report, (iv) *deactivation* of data shapes to exclude them from the validation process, and (v) *non-validating characteristics* to annotate data shapes.

2.2. Constraint Types

More than eighty constraint types were identified [21] from which a subset is used as axioms in ontologies [22] and a subset motivated the creation of SHACL [23]. Existing approaches to generate RDF constraints use UML diagrams or ontologies as source but usually cover only a limited subset of SHACL core constraint types due to an incomplete mapping. We count SHACL core constraint types based on the "Core Constraint Components" of SHACL specification [2].

The **Open Standards for Linking Organizations (OSLO)** initiative of Flanders, Belgium generates SHACL constraints annotated UML models [24] representing RDF classes and properties. The generated SHACL constraints are limited to a subset of constraint types, i.e. cardinality, class, and datatype, i.e. only support 3 out of 31 SHACL core constraint types.

Automatic Generation of SHACL Shapes from Ontologies (Astrea) [25] is based on a mapping of conceptual restrictions between patterns of OWL axioms and SHACL constraints. These patterns only contain 20 out of the 31 SHACL core constraint types when counting the core constraint types of the SHACL specification and not their parameterizations. For instance, we count the constraint type `sh:nodeKind` once and we do not count its parameterizations, such as "`sh:nodeKind sh:Literal`" or "`sh:nodeKind sh:BlankNode`". Besides these core constraints, Astrea also covers other concepts of the SHACL specification, namely *property paths* and terms related to *targeting* which applies elements of the shapes graph to elements of the data graph; we also support these concepts and additionally the concepts of *deactivation* and *severity* of data shapes.

TopQuadrant generated SHACL constraints from the **RDFa of the schema.org vocabulary**[3]. These constraints consist of the constraint types class, datatype and disjunction, i.e. only 3 out of 31 constraint types.

Manually created RDF constraints are theoretically not limited by any mapping as a user potentially can use all constraint types of a specification. However, similar to ontology axioms [22] only a subset seems to find common use. In our previous work [12] and later updated and extended statistics[4], we investigated the use of constraint types in SHACL shapes. We found that 30 out of 31 constraint types were used, but only a few are used in more than 60 percent of surveyed GitHub repositories: value type (class, datatype, nodekind), cardinality and disjunction constraints. Thus, RDF constraint visualizations and editors should *at least* cover these commonly used constraint types; however, to avoid a self-fulfilling prophecy where such a limitation reinforces the use of already commonly used constraint types, editors should not be limited to *only* these constraint types either.

2.3. RDF Constraint Editors

Tools to edit RDF constraints already exist but are either based on a specific textual syntax or have no formally defined visual notation.

Fajar et al. [26] implemented a **SHACL editor as plugin for Protégé**. However, their plugin is text-based and does not use a visualization for RDF constraints, therefore users are required to learn a specific RDF constraint language. Similarly, the tool **ShapeDesigner** from Boneva et al. [27] provides a text-based interface in which users are confronted with ShEx and SHACL representations of RDF constraints.

The tool **TopBraid Composer** from TopQuadrant can be considered as a SHACL editor[5]. It uses forms as a graphical interface for users, but, given it is commercial, no detailed specifications are available.

De Meester et al. [9] list features for a visual data shape editor implemented in an early version of the visual editor **UnSHACLed**. Although a few comments regarding the visualization were made, important details are not specified. For instance, the meaning of arrows or the selection of colors is not clearly specified, preventing developers of other tools from effectively implementing the visual notation. As a result, the original visualization of UnSHACLed is coupled to the tool hampering the accessibility for users across tools.

RDFShape [10] considers UML-like class diagrams. However, it does not cover all commonly used constraint types and, similarly to UnSHACLed, does not specify all details of how RDF constraints are visualized. The tool only statically visualizes RDF constraints and, currently, does not support logical relationships, e.g. (exclusive) disjunction[6], – commonly used according to preliminary statistics [12]. Even though support for additional constraint types can be implemented, it is not specified how it should be visualized, leaving room for different interpretations.

OntoPad[7] and **shaclEditor**[8] are visually editors for RDF providing a way to visually interact with SHACL shapes. Both editors are built on the QuitStore[9], a collaborative workspace for RDF datasets and use different visualizations which are not specified.

2.4. Semantic Web Visualizations

We look into the visualization of other Semantic Web concepts because they might be relevant for the visualization of RDF constraints.

UML is often used for modeling ontologies. The creation of constraints on RDF data from a conceptual point of view shows similarities to the creation of axioms in an ontology. Thus, visualizations for ontologies would be expected to be applicable to RDF constraints as well. A simple version of UML is used within the structural specification of OWL [28] to vi-

[3] http://datashapes.org/schema
[4] https://zenodo.org/record/4154456
[5] https://www.topquadrant.com/technology/shacl/tutorial/
[6] https://github.com/weso/umlShaclex/blob/06230fc568d0d91d443bb9ae819b9a1e65c6cc4e/src/main/scala/es/weso/uml/ShEx2UML.scala#L112
[7] https://aksw.github.io/OntoPad/
[8] https://github.com/firmao/shaclEditor
[9] https://github.com/AKSW/QuitStore

sualize the definition of conceptual restrictions in the form of axioms. Cranefield and Purvis [14] demonstrate how a subset of UML and the associated Object Constraint Language (OCL) [29] is used to model ontologies. Even the Object Management Group (OMG) – which maintains the UML specification – defined a specific UML profile for OWL and RDF, the Ontology Definition Metamodel (ODM) [15].

A plethora of ontology visualizations exists, but **VOWL** appears to be the most prominent visualization with respect to practical use and user familiarity for several concepts related to RDF constraints. Combining findings of several surveys [30–35] and two works [36, 37] presenting visualization tools, 84 ontology visualization tools were identified. *Widoco* [38], a widely used tool to create ontology documentations, uses *WebVOWL* [39] to visualize ontologies. WebVOWL implements the Visual Notation for OWL Ontologies (VOWL) [4]. VOWL is also implemented as a plugin for the commonly used modeling tool Protégé in ProtégéVOWL [40]. This suggests that users who use ontologies and read their documentations have at least encountered a VOWL-based visualization. Besides ontologies, VOWL-based visualizations also exist for queries [6], Linked Data visualization [7] and generation [5], all closely related to RDF constraints.

3. Visual Notations

We introduce two visual notations for RDF constraints to establish a baseline for a fair comparison, we provide general design considerations for both notations, *ShapeUML* (based on UML) and *ShapeVOWL* (based on VOWL). Both visualize fundamental constructs of RDF constraint languages: *constraints* and the context in which they are applied, i.e. *data shapes*. We describe which visual variables are used as graphical primitives for both notations, following Moody [8] and thus make design decisions transparent. Cognitive effective design principles [8] where taken into account where applicable, a detailed comparison between both notations based on these principles can be found in the next section (Section 4).

Both notations have different visual features and represent all SHACL core constraints and additionally concepts related to *targeting*, *property paths*, *severity* and *deactivation*; although both notations are built based on SHACL, they are constraint language independent and semantic constructs of other constraint languages can be mapped to it. Thus, both notations represent the same semantic constructs and their only difference are their visual features, enabling a fair comparison. Currently the visual notations visualize all SHACL core constraints, where necessary with (additional) constraint-language-independent text labels; Figs. 1, 2, 5 and 6 list all SHACL core core constraints and the other supported concepts together with a corresponding terminology mapping used by our notations *ShapeUML* and *ShapeVOWL*.

3.1. ShapeUML

The ShapeUML is based on the Ontology Definition Metamodel (ODM) [15] in which both nodes and properties are first-class UML constructs and, thus, graphically represented as class diagram boxes (rectangle). Therefore, constraints on both nodes and properties can be expressed and logical relationships between different types of data shapes can be visualized.

The graphical primitives of *ShapeUML* are the following visual variables [8]: shape, edge, text, border and position. The full specification is available at https://w3id.org/imec/unshacled/spec/shape-uml/20210118/. In the remaining, we describe the graphical primitives and elaborate with an example.

3.1.1. Shape

We **reuse classes (rectangles)** from UML [13] to represent both node and property shapes, **redefine the meaning of rectangle's compartments** for RDF constraint specifics, **introduce data shape stereotypes** to indicate a data shape's type and distinguish it from other UML rectangles representing other concepts.

We use the graphical primitive *shape* to represent the fundamental construct *data shapes* and its subclasses *node* and *property shape* thus adhering to ODM [15]. *Data shapes* are represented using a **rectangle** (Fig. 3 (1)), and describe constraints applying on subjects and objects from the data graph. *Node shapes* describe constraints on individual focus nodes, while *property shapes* describe constraints for reachable nodes via a property path.

In UML "a class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines" [13] which we redefine for *data shapes*. The **upper compartment** contains the *data shape*'s type and name (Fig. 3 1). We determine the *data shape*'s type by reusing UML concepts similar to the UML profile for OWL and RDF [15], i.e. we define UML "stereotypes" to signify what the rectangles represent: *node shapes* declared as «NodeConditions», property

Category	SHACL term	Text dispayed at {PLACEHOLDER}
Value Type	sh:class	class ex:testClass
	sh:datatype	datatype ex:testType
	sh:nodeKind	nodeKind IR)
Cardinality	sh:minCount	
	sh:maxCount	
Value Range	sh:minExclusive	minExclusive 5
	sh:minInclusive	minInclusive 5
	sh:maxExclusive	maxExclusive 10
	sh:maxInclusive	maxInclusive 10
String-based	sh:minLength	minLength 5
	sh:maxLength	maxLength 10
	sh:pattern	pattern /p/flags
	sh:flags	
	sh:languageIn	languageIn ('en')
	sh:uniqueLang	uniqueLang true
Property pair	sh:equals	equals ex:test
	sh:disjoint	disjoint ex:test
	sh:lessThan	lessThan ex:test
	sh:lessThanOrEquals	lessThanOrEquals ex:test
Logical	sh:not	
	sh:and	AND
	sh:or	OR
	sh:xone	OneOf
Shape-based	sh:node	
	sh:property	
	sh:qualiviedValueShape	
	sh:qualifiedMinCount	
	sh:qualififedMaxCount	
Other	sh:closed	onlyListedProperties true
	sh:ignoredProperties	otherAllowedProperties ex:test
	sh:hasValue	hasValue ex:test
	sh:in	valueIn (ex:test1, ex:test2)

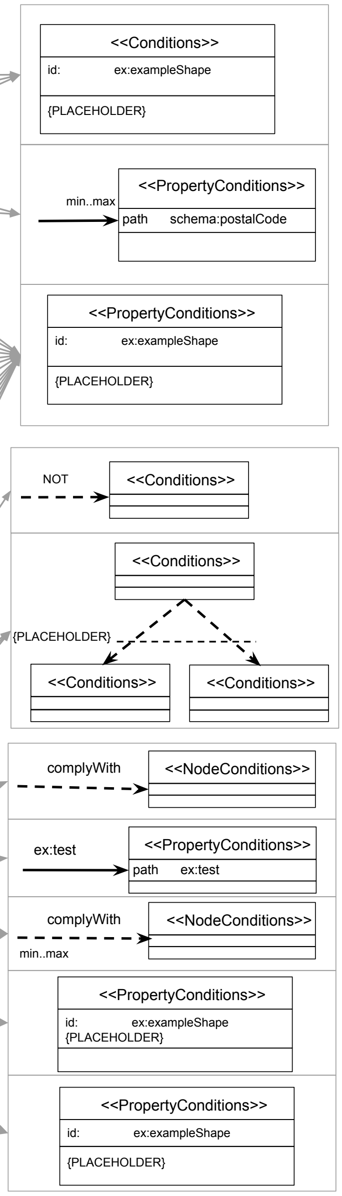

Fig. 1. Correspondence between semantic constructs and *ShapeUML*: SHACL core constraints (left) and graphical notations (right).

shapes declared as «PropertyConditions» and (if the data shape type is not specified) data shapes as «Conditions». The name of the data shape is displayed as bold text to support the user in the identification and differentiation of data shapes. This name may be populated from `rdfs:label` values of the data shape, thus following best practices in labeling RDF concepts for humans. Both the middle and lower compartment list text-based key-value pairs, therefore we stay compliant to *UML*. Additionally, constraint language independent labels (Figs. 1 and 2) are used to convey meaning and support users. The **middle compartment** lists information about the *data shape*'s identification and validation (Fig. 3 7). Thus, *data shapes* are similar to UML where the middle compartment usually contains the *attributes* of *classes*, i.e. what characterizes them. The **lower compartment** contains actual *constraints* as a key-value list (Fig. 3 3).

Category	SHACL term	Text displayed at {PLACEHOLDER}
Target	sh:targetNode	appliesOn(ex:alice)
	sh:targetClass	appliesOn class(ex:Test)
	sh:targetSubjectsOf	appliesOn subjectsOf(ex:Test)
	sh:targetObjectsOf	appliesOn objectsOf(ex:Test)
Severity	sh:severity	severity Warning
Deactivation	sh:deactivated	deactivated true
Property Path	Predicate path	ex:parent
	sh:inversePath	^ex:parent
	Sequence path	ex:parent/ex:firstName
	sh:alternativePath	ex:father | ex:mother
	sh:zeroOrMorePath	rdfs:subClassOf*
	sh:oneOrMorePath	rdfs:subClassOf?
	sh:zeroOrOnePath	rdfs:subClassOf+

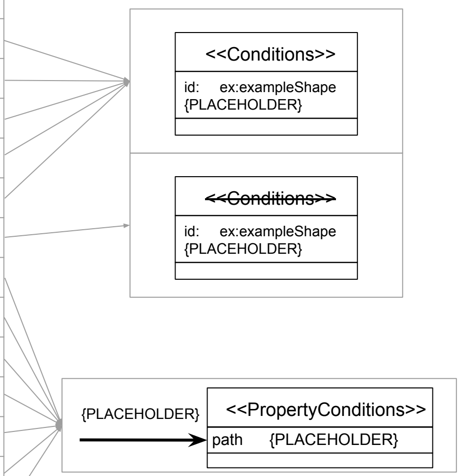

Fig. 2. Correspondence between semantic constructs and *ShapeUML*: other relevant SHACL concepts besides core constraints (left) and graphical notations (right).

3.1.2. Edge

We **reuse directed solid edges** from ODM/UML [15] to represent relationships, **reuse dashed edges overlaying individual edges** from UML [13] to represent one-to-many relationships, and **redefine directed dashed edges** for RDF constraint specifics.

Directed edges represent different relationships between data shapes and, thus, *ShapeUML* is able to represent relationships between different types of *data shapes*. Directed edges have a *label* at the **center of the edge** and possibly *cardinalities* next to the ends of the association (Fig. 3 2). These edges associate a *data shape* with another *data shape* or set of *data shapes*.

We introduce **solid** and **dashed** directed edges to visually distinguish between different types of relationships. We indicate the edges from *node shapes* to *property shapes* as a **directed solid edge** (Fig. 3 2) as it represents relationships between subjects and objects of the data graph. The *label* of such a connection is the property path of the connected *property shape* which supports readability as humans can read the label while processing the edge and do not have to look for this label elsewhere in a rectangle; annotating an edge with a label also follows UML. A **dashed directed edge** with the label *complyWith* indicates that the source *data shape* needs to comply with the constraints of the destination *data shape* (Fig. 3 5). Therefore such connections can be distinguished from property shape connections both via a visual difference and a different label. Similarly, a dashed directed edge with the label *NOT* indicates that the source *data shape* must not comply with the destination *data shape* (Fig. 3 6). A **dashed line vertically** over individual edges with label next to the dashed line indicates one-to-many relationships between a *data shape* to a set of *data shapes*, following the UML specification [13] (Fig. 3 5).

3.1.3. Text

We **reuse text** from UML to represent different concepts and **introduce striked through text** for data shape stereotypes to indicate a deactivated data shape.

Text represents *constraints* stated by a *data shape* and provides additional information where necessary. Text is added to the upper, middle and lower compartment of a *data shape* and as label on edges. The type of a data shape in the upper compartment can be striked through, showing that the *constraints* of this *data shape* are not used for validation, i.e. the data shape is deactivated (Fig. 3 8). This visual aid aims in the quick identification of deactivated *data shapes* which does not introduce any visual symbol and thus does not deviate too much from the *UML* specification.

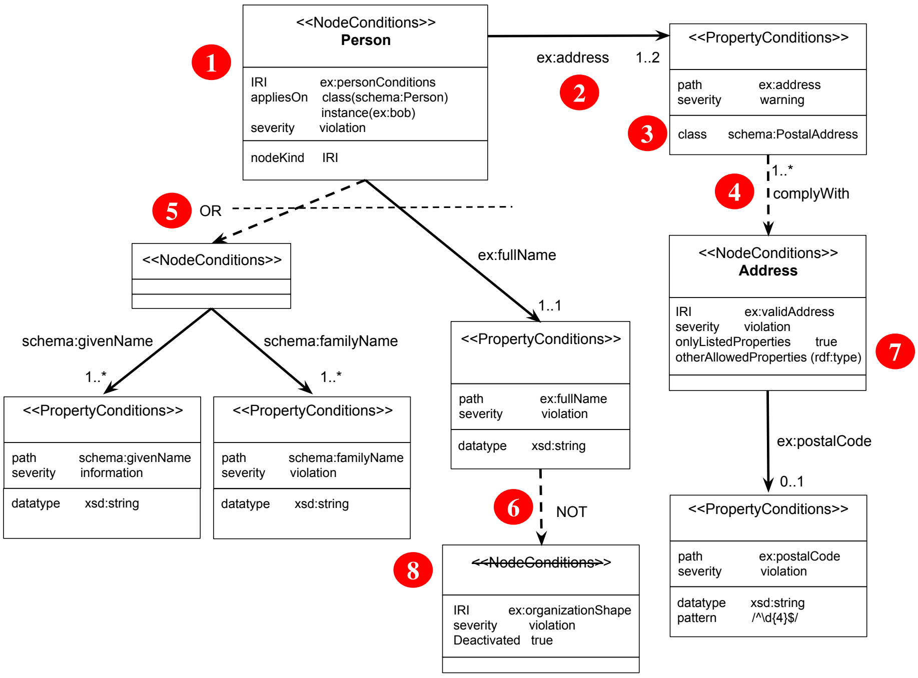

Fig. 3. Constraints visualized using ShapeUML: A subject valid to the Person data shape should have an IRI (1), at least one but maximum two `ex:address` properties (2) of class `schema:PostalAddress` (3) and the object of at least one `ex:address` property should comply with the existing data shape `ex:validAddress` (4). Additionally, the subject valid to person should either have exactly one `ex:fullName` or at least one `schema:givenName` (5) and at least one `schema:familyName` all of datatype `xsd:string`. The value of `ex:fullName` must not comply with the data shape `ex:organizationShape` (6). Addresses must only have values for the property `postalAddress` with an exception for `rdf:type` (7). Constraints of the `ex:organizationShape` are not considered for validation (8).

Values referring to RDF terms can be shortened with a prefix, therefore the tool implementing the visual notation has to provide a prefix list.

3.1.4. Border

We **reuse solid borders** from UML, they are used for *data shapes*. According to the UML standard, stylistic details, such as line thicknesses, are not material to the specification. So, all *data shapes* are rendered using solid borders.

3.1.5. Position

We **reuse positions at the beginning and end of directed edges** from UML to represent cardinality-related constraint types. Within UML, *association ends* are among others specified by their cardinality.

In ShapeUML, cardinality constraints referring to properties are visualized next to the arrow head of a directed edge, i.e. *minCount* and *maxCount* (Fig. 3 2); cardinality constraints referring to data shapes are visualized next to the source of a directed edge, i.e. *qualifiedMinCount* and *qualifiedMaxCount* (Fig. 3 4). Thus, the visualization reflects the reading direction, for example: the person data shape requires the property `ex:address` at least 1 but maximum 2 times (Fig. 3 2) vs a valid address property requires that at least 1 property value need to comply with the address node shape (Fig. 3 4).

3.1.6. Visual Example

The visual vocabulary of *ShapeUML* defined in the last section, can be used to represent SHACL shape graphs. We present and discuss an example (Fig. 3).

ShapeUML defines visual elements for data shapes (Fig. 3). *Data shapes* of different types («Conditions», «NodeConditions» and «PropertyConditions») can be uniquely identified with an IRI but can also have a human readable label. For example, a *node shape* uniquely identified (`ex:personConditions`, middle compartment) can have the human readable name *Person* (bold label in upper compartment) (Fig. 3 1). Such a *node shape* can by default be applied on resources, e.g. `ex:bob`, or all instances of a class, e.g. `schema:Person`, both indicated by the key *appliesOn* in the middle compartment of a *ShapeUML data shape*.

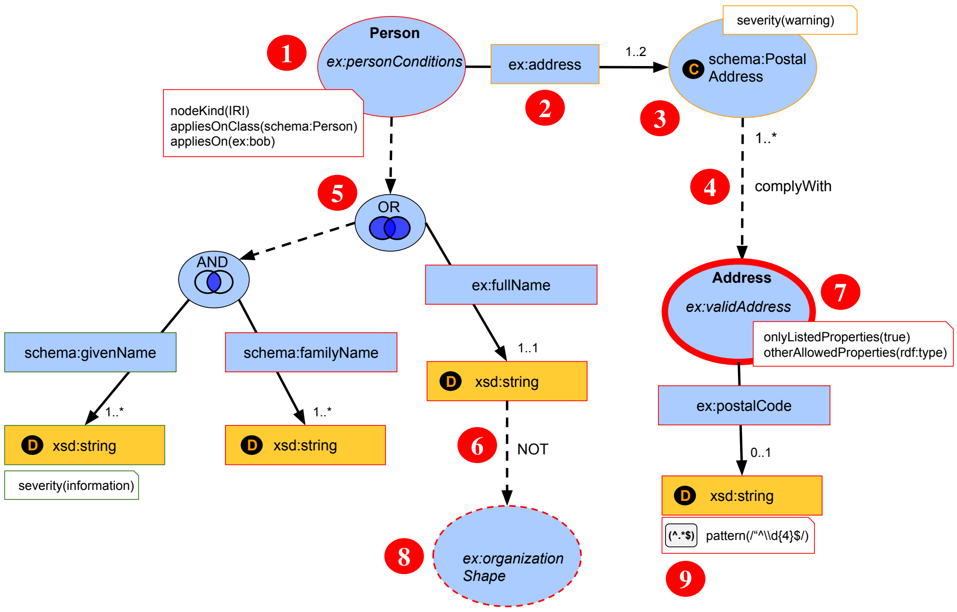

Fig. 4. Constraints expressed using ShapeVOWL: A subject valid to the Person data shape should have an IRI (1), at least one but maximum two `ex:address` properties (2) of class `schema:PostalAddress` (3) and the object of at least one `ex:address` property should comply with the existing condition set `ex:validAddress` (4). Additionally, the subject valid to person should also either have exactly one `ex:fullName` or at least one `schema:givenName` (5) and at least one `schema:familyName` all of datatype `xsd:string`. The value of `ex:fullName` must not comply with the data shape `ex:organizationShape` (6). Addresses must only have values for the property `postalAddress` with an exception for `rdf:type` (7). Constraints of the `ex:organizationShape` are not considered for validation (8). ShapeVOWL also visualizes optional accompanying logos for constraint types (9).

Constraints are listed in the lower compartment of a *data shape* rectangle. A node could be constrained to be of a specific type using the *nodeKind* constraint. Similarly, constraints on property values are placed in the lower compartment of the corresponding «PropertyConditions» *property shape*. A fictive person node shape can represent the constraint that data valid to this *data shape* must have a unique identifier. And in the same fashion, the value of an `ex:address` property can be constrained to be of a specific class (Fig. 3 3).

Cardinality constraints are represented using *text* and *position*. Therefore a constraint to express that a person must have at least *one* but maximum *two* addresses will be denoted with the (inclusive) cardinality specification `1..2` next to the arrow head of the directed edge which connects the *person node shape* with the *address property shape* (Fig. 3 2).

Dashed directed edges can be used to indicate reuse of data shapes. To denote that the value of *at least one* of the aforementioned `ex:address` properties must comply with the `ex:validAddress` data shape, a dashed relationship with corresponding cardinalities `1..*` is drawn at the source *property shape* (Fig. 3 4). In case every *address* should comply with the provided data shape, the qualified cardinalities at the source of the dashed arrow need to be removed. Such a removal would mean for a SHACL implementation that the two constraints `sh:qualifiedValueShape` and related `sh:qualifiedMinCount` are replaced by a single `sh:node` constraint. However, this is transparent in the visualization and users are not bothered with this specific terminology.

Data shapes can be connected with logical operators to build more complex constraints (Fig. 3 5): subjects valid to the *Person node shape* should have either *exactly* one `ex:fullName` property, or at least one `schema:givenName` and at least one `schema:familyName`: dashed vertical OR edge overlaying individual edges.

3.2. ShapeVOWL

This visual notation is based on *VOWL* [4] and designed to be as close as possible to it. The graphical primitives of *ShapeVOWL* are shape, edge, text, border, position and color. The full specification is available online at https://w3id.org/imec/unshacled/spec/shape-vowl/20210118 We describe the graphical primitives and elaborate with an example.

3.2.1. Shape

We **reuse blue ellipses and blue and yellow rectangles** from VOWL to represent subjects, predicates and objects of the data shape graph and **introduce white note-elements** to represent constraints.

The graphical primitive *shape* distinguishes the fundamental constructs *node shapes*, *property shapes* and *constraints*, and represents one-to-many relationships. This follows *VOWL* where nodes in the graphs as well as specific restrictions such as *disjointness* are represented with dedicated nodes. *Node shapes*, subjects of triples, are represented as **ellipses** (Fig. 4 1), *property shapes*, the predicate and object of a triple, as **rectangular label** on a directed edge (Fig. 4 2) and either a **ellipse** or **rectangle** at the end of the edge (Fig. 4 3, 6), and *constraints* as **rectangle with the upper right corner bent** (note element) (Fig. 4 1, 6). Thus, node and property shapes align with VOWL as the *data shapes* appear like the RDF graph on which they define constraints on.

The note-element, containing constraints as text, is visually attached at the *node shape* or *property shape* indicating the constraints applying on the represented subjects, predicates or objects of a triple; constraints are visualized where they apply to facilitate the processing of the visualization by users. We also introduce ellipses as intermediate element to denote one-to-many relationships (see edges).

3.2.2. Edge

We **reuse directed solid edges with rectangular labels** from VOWL to represent properties and **redefine directed dashed edges** for RDF constraint specifics.

Edges represent relationships between *data shapes* which makes *ShapeVOWL* able to represent different kind of constraints in a visual fashion. **Directed dashed edges** (Fig. 4 4) refer to relationships between *data shapes* and denote their label directly as text on top of the relationship. They indicate that the source *data shape* needs to comply with the constraints of the destination *data shape*.

Directed solid edges are part of a *property shape* and indicate their *label* in a **rectangle** above the edge (Fig. 4 2), following *VOWL*. The *label* of directed solid edges is the property path of the represented property shape; relationships between *data shapes* are visually distinguished from *property shapes* due to the use of different edges.

Similar to VOWL, *cardinalities* are denoted next to the arrow head (Fig. 4 2), but additionally *data shape* related qualified cardinalities are denoted at the start of a directed dashed edge (Fig. 4 4). *Node and property shapes* may refer to multiple other *node and property shapes* in a **one-to-many relationship** to represent logical relationships. We represent such relationships using additional ellipses, representing the meaning of individual one-to-many relationship, i.e. conjunction and (exclusive) disjunction (Fig. 4 5), similar to certain restrictions in *VOWL*, e.g., *disjointness*.

3.2.3. Text

We **reuse text** from VOWL to represent labels, **redefine datatype** to represent datatype constraints, **introduce text** to represent constraints and **italic text** to represent the unique identifer of data shapes.

We use **text** to represent constraints stated by *data shapes*, unique identifiers, and labels. Text is added in constraint *note elements*, *node shapes* and *property shapes*. Constraint note elements contain *constraints* in the form of text where the constraint's name is listed followed by its value in parentheses. This allows a consistent representation of different constraints without introducing a new visual variable for each of possibly more than 80 constraint types [21]. Values referring to RDF terms can be shortened with a prefix, therefore the tool implementing the visual notation has to provide a prefix list. *Data shapes* may have an optional human readable name which is denoted as bold text in the upper part of the *data shape* to facilitate the distinction of data shapes. This name may be populated from `rdfs:label` values, and, thus following best practices for labeling RDF concepts. Additionally, the unique identifier of *node shapes* is visualized as text in italics in the center of the ellipse representing the *node shape* (Fig. 4 1). Users can also identify *node shapes* without a human readable label present. The *italic* type distinguishes the unique identifier from other text.

3.2.4. Border

All visual shapes have a border, we **reuse solid borders** from VOWL, **redefine dashed borders** to accommodate for validation-specific characteristics regarding deactivation and **introduce thick solid borders** to represent the constraint type *closed*.

VOWL uses dashed borders for specific OWL classes and literals without datatype. However, we use **dashed borders** to indicate which *data shapes* are not considered for validation (*deactivated*), because in contrast to an ontology visualization, we do not consider specific OWL classes but RDF constraints for

Category	SHACL term	Optional icon and text dispayed at {PLACEHOLDER}	
Value Type	sh:class	C	ex:testClass
	sh:datatype	D	ex:testType
	sh:nodeKind		nodeKind(IRI)
Cardinality	sh:minCount		
	sh:maxCount		
Value Range	sh:minExclusive		range(min..max)
	sh:minInclusive		
	sh:maxExclusive		
	sh:maxInclusive		
String-based	sh:minLength		length(min..max)
	sh:maxLength		
	sh:pattern	(^.*$)	pattern(/p/flags)
	sh:flags		
	sh:languageIn		languageIn(en)
	sh:uniqueLang		uniqueLang(true)
Property pair	sh:equals	=	equals(ex:test)
	sh:disjoint	≠	disjoint(ex:test)
	sh:lessThan	<	lessThan(ex:test)
	sh:lessThanOrEquals	≤	lessThanOrEquals(ex:test)
Logical	sh:not		
	sh:and		
	sh:or		
	sh:xone		
Shape-based	sh:node		
	sh:property		
	sh:qualiviedValueShape		
	sh:qualifiedMinCount		
	sh:qualififedMaxCount		
Other	sh:closed		onlyListedProperties(true)
	sh:ignoredProperties		otherAllowedProperties(ex:test)
	sh:hasValue		hasValue(ex:test)
	sh:in		valueIn(ex:test1, ex:test2)

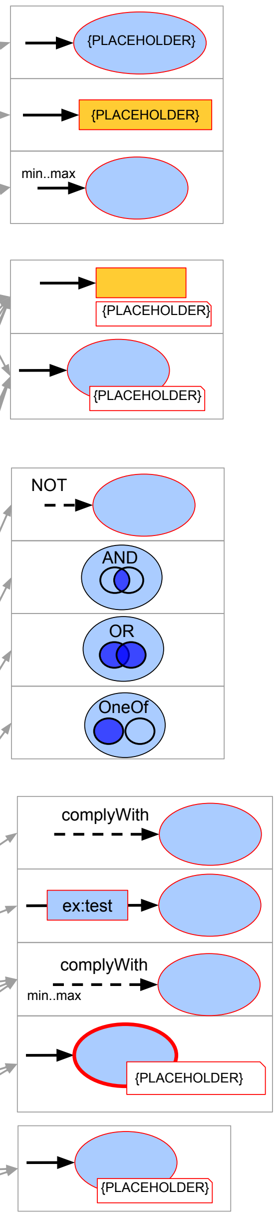

Fig. 5. Correspondence between semantic constructs and *ShapeVOWL*: SHACL core constraints (left) and graphical notations (right).

validation, and our visualization of literals has a different meaning as we visualize constraints (Fig. 4 9). For deactivated *node shapes* both the ellipse representing the *node shape* as well as a possibly attached note element with constraints will get a dashed border (Fig. 4 8). Similarly, for deactivated *property shapes* the rectangle of the relationship label, the object and potentially attached note elements get a dashed border.

We introduce **thick solid borders** for *node shapes*, indicating that for validation only the explicitly linked properties are allowed (*closed data shape*, Fig. 4 7).

The thick borders aim to represent the closeness whereas dashed borders aim to represent inactiveness. As the thickness and style of the edges are two different visual features, possible combinations of deactivated and closed data shapes can still be represented.

Category	SHACL term	Text displayed at {PLACEHOLDER}
Target	sh:targetNode	appliesOn(ex:alice)
	sh:targetClass	appliesOnClass(ex:Test)
	sh:targetSubjectsOf	appliesOnSubjectsOf(ex:Test)
	sh:targetObjectsOf	appliesOnObjectsOf(ex:Test)
Severity	sh:severity	severity(Warning)
Deactivation	sh:deactivated	deactivated(true)
Property Path	Predicate path	ex:parent
	sh:inversePath	^ex:parent
	Sequence path	ex:parent/ex:firstName
	sh:alternativePath	ex:father | ex:mother
	sh:zeroOrMorePath	rdfs:subClassOf*
	sh:oneOrMorePath	rdfs:subClassOf?
	sh:zeroOrOnePath	rdfs:subClassOf+

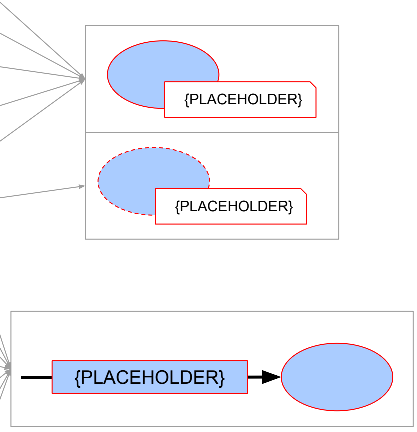

Fig. 6. Correspondence between semantic constructs and *ShapeVOWL*: other relevant SHACL concepts besides core constraints (left) and graphical notations (right).

3.2.5. Position

We **reuse cardinality positions at directed edge endings** for property-based cardinality constraints, **introduce cardinalities at the beginning of a directed edge** to represent data shape related cardinality constraints, **introduce positions for logical constraints** within dedicated nodes and **introduce positions for datatype and class constraints** within the objects of visualized triples.

We use specific **positions** for cardinality, datatype, class and logical constraints utilizing the graph visualization to support users in the parsing of information. In *ShapeVOWL*, cardinality constraints referring to properties are visualized next to the arrow head of a directed edge, i.e. *minCount* and *maxCount*; cardinality constraints referring to data shapes are visualized next to the source of a directed edge, i.e. *qualifiedMinCount* and *qualifiedMaxCount* (Fig. 4 4). The visualization reflects the reading direction, for example: the person data shape requires the property `ex:address` at least 1 but maximum 2 times (Fig. 4 2) vs a valid address property requires at least 1 property value to comply with the address node shape (Fig. 4 4).

Datatype and class constraints are not visualized in a note element, but directly as text in the graphical element representing the object, i.e. a yellow rectangle for datatype constraints (Fig. 4 6) or a blue ellipse for class constraints (Fig. 4 3). VOWL visualizes datatypes as text within the yellow rectangle representing a literal. We reuse this visualization to denote a datatype constraint of a property value and add an additional datatype icon in front of the name of the datatype to indicate that a constraint exists (Fig. 4 6). This icon is an orange D in a black circle (Fig. 4). Consistently with datatypes, class constraints are denoted as text within the ellipse representing the property value. Class constraints have an additional class icon in front of the name of the class. This icon is an orange C in a black circle (Fig. 4 3).

Logical constraints are not represented in a note element, but as dedicated nodes or as labels on dashed edges which enables *ShapeVOWL* to represent relationships between different types of *data shapes*. Conjunction and (exclusive) disjunction constraints are visualized as ellipse with respective labels on the upper part of the ellipse (Fig. 4 5). Additionally, icons representing Venn diagrams are used to distinguish the different logical constraint types. These icons represent Venn diagrams, similar to certain VOWL constructs. Negation constraints are represented as text label "NOT" on top of dashed edges connecting data shapes (Fig. 4 6).

3.2.6. Color scheme

We **reuse the VOWL base color** to represent subjects, predicates and objects of the data shape graph, **reuse the VOWL literal color** to represent literals and **introduce border colors** for data shapes' severity.

A color scheme is applied on the border color of *data shapes* and *note elements* to express different severities (Fig. 4 1). VOWL uses a color scheme for a better distinction of the different elements [4]. We reuse the base color and literal color of VOWL.

Additionally, for *ShapeVOWL* colors on borders are used to express the severity of *data shapes*. For the severities *violation, warning and information* from the SHACL specification we recommend the respective colors **red, yellow and green**. Green is chosen instead of blue so the severity colors for *data shapes* are not confused with the VOWL *general color*.

3.2.7. Visual Example

The visual vocabulary of *ShapeVOWL* defined in the last section, can be used to represent SHACL shape graphs. We present and discuss an example (Fig. 4).

ShapeVOWL defines visual elements for data shapes (Fig. 4). Our color scheme is applied; *data shapes* are colored with respect to their severity.

Node shapes can be uniquely identified with an IRI but can also have a human readable label. For example, a *node shape* uniquely identified with the IRI `ex:personConditions` (center of ellipse representing a subject node) can have the human readable name *Person* (bold label in upper part of the ellipse) (Fig. 4 1). Such a *node shape* can by default be applied on resources, e.g. `ex:bob` or all instances of a class such as `schema:Person`, both is is indicated by the *appliesOn* annotation in the attached white note-element of a *ShapeVOWL data shape*.

Constraints have a special position or are listed in white note-elements attached to a *data shape*; depending on the rendering either overlapping an ellipse (Fig. 4 1) or next to a rectangle (Fig. 4 9). A fictive person node shape can represent the constraint that persons must have a unique identifier (Fig. 4 1, nodeKind constraint). The value of an `ex:address` property can be constrained to be of a specific class whereas value type constraints are listed within the shape representing the object together with an icon (Fig. 4 3). Cardinality constraints are represented using *text* and *position*. Thus, a constraint to express that a person must have at least *one* but maximum *two* addresses will be denoted with the (inclusive) cardinality specification `1..2` next to the arrow head of the directed edge which connects the *person node shape* with the *address property shape* (Fig. 4 2).

Dashed directed edges with the label *complyWith* indicate **reuse of data shapes**. To denote the constraint that the value of *at least one* of the aforementioned `ex:address` properties must comply with the `ex:validAddress` data shape, a dashed relationship with corresponding cardinalities `1..*` is drawn at the source *property shape* (Fig. 4 4). In case every *address* should comply with the provided data shape, the qualified cardinalities at the source of the dashed arrow have to be removed.

Data shapes can be connected with logical operators to build more complex constraints (Fig. 4 5): subjects valid to the *Person node shape* should have either *exactly* one `ex:fullName` property, or at least one `schema:givenName` and at least one `schema:familyName`: disjunction node with label "OR" and Venn diagram icon. The logical operator negation only takes one data shape as argument and not a whole data shape list, therefore it is visualized with the label NOT on a dashed connection (Fig. 4 6).

With respect to validation **data shapes may be closed or deactivated**. The `ex:validAddress` data shape is closed, visually indicated by a thick border: valid addresses are only allowed to have the property `postalCode` and an exception is made for `rdf:type` denoting the class (Fig. 4 7). The data shape `ex:organizationShape` is deactivated, visually indicated by dashed border: its constraints are not considered during validation (Fig. 4 8). Constraint types can be accompanied with a logo displayed before the constraint in the note element (Fig. 4 9).

4. Comparative Analysis

Both *ShapeUML* and *ShapeVOWL* were designed by following basic principles of cognitive effectiveness [8], however, as we reused the existing notations *UML* and *VOWL* these principles could only be applied to a certain extent. Therefore, we analyze *ShapeUML* and *ShapeVOWL* with respect to these design principles with the aim of scientifically argue about the impact of design decisions on human information processing and thus the effectiveness of *ShapeUML* and *ShapeVOWL* from a theoretical perspective.

We refer to each principle's definition according to Moody [8] and discuss to which extent each visual notation complies. We omit the design principle *cognitive integration* as it only applies when multiple diagrams of different types are integrated. Table 1 summarizes the comparison which is discussed in Section 4.9.

4.1. Semiotic Clarity

Semiotic clarity relates to the correspondence between symbols and their referent concepts [8]. In case of *symbol redundancy*, a semantic construct is represented by multiple graphical symbols; the opposite is *symbol overload*. *Symbol excess* occurs if graphical symbols do not correspond to any semantic construct; and the opposite is *symbol deficit*, a semantic construct with no graphical symbol.

ShapeUML All semantic constructs are represented in the visual notation (Figs. 1 and 2), i.e. terms from the SHACL specification; some constructs use the same graphical symbol but text is used to differentiate, and, thus, to maintain visual expressiveness. Following the ODM-profile of UML, *ShapeUML* uses rectangles with solid borders to represent *data shapes*, thus *node* and *property shapes* share the same graphical symbol (**symbol overload**). However, *node* and *property shapes* are distinguished by additional text indicating the type. **Symbol deficit** was deliberately introduced to reduce graphic complexity: more than 30 constraint types are supported, but they are all represented as text, only *logical constraint types* and *cardinality constraints* use additional visual variables (*edges* and *position*). *ShapeUML* does not visualize any semantic construct with multiple graphical symbols (*symbol redundancy*) nor does it contain any graphical symbol which does not correspond to a semantic construct (*symbol excess*), thus semiotic clarity is achieved.

ShapeVOWL All semantic constructs are represented in the visual notation (Figs. 5 and 6) and similar to ShapeUML, **symbol deficit** is deliberately introduced to increase visual expressiveness. Multiple graphical symbols are used in ShapeVOWL. Circles represent *node shapes* (subject of triples) but also part of *property shapes* (objects of triples). However, as *node shapes* are represented as subjects, they can be distinguished from objects because they only have outgoing solid edges with property paths in a rectangular label; ingoing edges are limited to dashed edges which indicate *node shape* reuse.

Certain constraint types are represented using the visual variables *border*, *edge* and *position* but to reduce graphic complexity most of the 31 constraint types are represented textually within *note-elements*. However, constraint types may also be accompanied by an icon which we provide for commonly used constraint types [22] which do not already are visualized using other visual variables such as *position* (see next section). Similar to *ShapeUML*, *ShapeVOWL* achieves *semiotic clarity* as no *symbol redundancy* nor *symbol excess* are present.

4.2. Perceptual Discriminability

Perceptual discriminability describes the ease and accuracy with which graphical symbols can be differentiated from each other [8]. A factor is the *visual distance*, i.e. the number of visual variables on which the symbols differ and the size of differences in perceptible steps (capacity). Shapes are the *primary basis* for humans to identify objects in the real world, while *textual differentiation* is a cognitively ineffective way to handle graphic complexity [8].

ShapeUML *ShapeUML* uses the visual variables shape, edge, text, border, and position. However, the **perceptual discriminability is low** as only one kind of shape and two types of edges are used. However, therefore we stay close to the UML specification, where users potentially are familiar with. Given the limited number of graphical symbols, i.e. rectangles with solid borders for *data shapes*, text for *constraints* as well as solid and dashed edges to relate *data shapes*, ShapeUML only provides **limited visual distance**.

ShapeVOWL *ShapeVOWL* uses the visual variables shape, edge, text, border, position, and color, thus **one visual variable more than *ShapeUML***. Nodes and properties are clearly distinguished by the visual variable *shape* and *color*, i.e. the VOWL base-color *blue* is used for nodes and property labels and the VOWL color *yellow* is used for literals. Additionally, the **visual distance between symbols is increased** because *ShapeVOWL* defines optional icons for different constraint types. Both subjects and potential objects are represented using ellipses. As discussed for *semiotic clarity*, this is *not* a case of *symbol overload* because *node* and *property shapes* can still be distinguished by the type of ingoing edges and whether it is a subject or object. However, this is a rather subtle difference with a low visual distance, thus perceptual discriminability is slightly decreased.

4.3. Semantic Transparency

Semantic transparency is the extent to which a notation's meaning can be inferred from its appearance, in-formally its "*intuitiveness*" or the degree of how much the appearance provides a cue to its meaning [8].

ShapeUML *ShapeUML* is based on UML which uses abstract shapes, and, thus it does **not provide much semantic transparency**. The boxes representing *data shapes* do not provide a cue to their meaning. However, presenting the property path as a label on edges connecting *node* with *property shapes* may resemble the underlying graph structure of RDF and could minimally provide *semantic transparency*.

ShapeVOWL *ShapeVOWL* uses a graph visualization based on nodes and edges of the actual RDF graph for which it defines the *constraints*. Several indicators suggest that *ShapeVOWL* has **high semantic transparency**. Previously defined VOWL-based visual notations already demonstrated that users find the graph visualization **intuitive** [4]. *ShapeVOWL* also reuses visual metaphors such as Venn diagrams for logical constraints, which, according to Moody, **increases semantic transparency**. *ShapeVOWL* attaches constraints visually to where they apply to which further increases semantic transparency; certain property shape constraints apply on the property, such as cardinalities, and others on the value of the property, such as *minimum inclusive value constraints*. If not visually separated, *min/max cardinality constraints* on the property and *min/max constraints* on the value might be confused. To further increase *semantic transparency*, *ShapeVOWL* defines optional icons for constraint types which can speed up recognition and recall as well as improve understanding for novice users [8].

4.4. Complexity Management

Complexity management aims not to overload the human mind. For instance, visual representations often do not scale well [8]. *Modularization* and *hierarchy* offer solutions to manage complexity.

Both proposed visual notations **do not yet account for modularization or hierarchy**. However, tools implementing visual notations can account for this and e.g. offer zoom functionality [5]. Currently our tool *UnSHACLed* provides geometric zooming (Section 5).

4.5. Visual Expressiveness

Visual expressiveness refers to the number of visual variables in the whole notation. Each variable has a power denoting the information which can be used [8].

The visual expressiveness of both visual notations **is not very high** considering that most *constraints types*, one of the fundamental constructs are represented as text only (with the exception of logical relationships in both notations). However, one the one hand this is because both notations were built with the **objective to reuse existing notations** already familiar to users, thus inheritance of *visual expressiveness*, and on the other hand we tried to **keep the graph complexity low** by deliberately not representing each constraint type with different visual variables.

If required by specific use cases, both notations can be improved specifically towards *visual expressiveness*. For example, **ShapeVOWL has higher expressiveness** due to the use of more visual variables compared to *ShapeUML*, in a similar fashion more visual variables can be used for both notations.

4.6. Dual Coding

Dual coding is the use of text to complement graphics. Text on its own is cognitively ineffective to encode information, but, in a supplementary fashion, it can reinforce and clarify meaning [8]. However, although *textual annotations* improve understanding, having a dedicated graphical symbol only for annotations not representing any semantic construct of the language it harms semiotic clarity, i.e. a case of *symbol excess* as the graphical symbol of annotation does not represent a semantic construct [8].

ShapeUML is based on UML, heavily text-based and thus **has limited dual coding**. Text is mostly used to denote constraints, but also for labels and unique identifiers. The *deactivation* of *data shapes* can be considered *dual coded* because, in addition to the textual declaration, the type of the *data shape* in the upper compartment is strikethrough, i.e. an additional visual change of font. *Node shapes* may refer to *property shapes* which in *ShapeUML* is encoded using a directed solid edge.

Following UML, *logical constraints* are represented with specific edges additionally labeled with the logical constraint's name. However, this is not considered *dual coded* as without label, edges of different *logical constraints* are not distinguishable. Both visual variable and text are needed to denote logical constraints.

ShapeVOWL visualizes graphs, and text is added to graph elements. **Several elements are dual coded** in *ShapeVOWL*. Similar to *ShapeUML*, text is mostly used to denote constraints, but also for labels, unique identifiers. All *constraints* are represented textually in a *note-element*, but some constraint types are also represented using additional icons or the visual variables *border*, *edge* and *color*. *ShapeVOWL* defines optional *icons* for constraint types, e.g. for class, datatype or literal pattern constraints. Together with the visual variable *color* and *border*, text also denotes the severity of data shapes. Dashed and thick solid borders, in addition to text, are used to indicate characteristics relevant to validation of the RDF constraints, the constraint type *closed* and deactivation of *data shapes*.

4.7. Graphic Economy

Graphic economy states that the size of the visual vocabulary should be cognitively manageable to achieve a low graphical complexity [8]. The *number of semantic constructs* can be limited, symbol deficit can be introduced or the visual expressiveness can be increased.

Both visual notations should be **cognitively manageable**. SHACL supports a subset of possibly more than eighty constraint types, thus the number of semantic constructs is already limited (all concepts listed in Figs. 1, 2, 5 and 6). Additionally, symbol deficit is deliberately introduced by the design decision of not visualizing each constraint type of the SHACL core using separate visual variables. An unlimited number of symbols can be created by combining visual variables, however, this does not scale due to cognitive limits where humans must remember the meaning of the symbol [8]. Both *ShapeUML* and *ShapeVOWL* have a small visual vocabulary as both use less than five graphical primitives.

4.8. Cognitive Fit

Cognitive fit means different representations are suitable for different tasks and audiences [8]. Optimizing visual notations for novice users can reduce effectiveness for experts and vice versa. More, the medium on which a visual notation is presented influences the effectiveness, i.e. manual drawing with pen and paper vs computer display. Icons, color, and texture are more difficult to draw than simple geometric shapes [8].

ShapeUML *ShapeUML* is based on UML, and, thus **is suited for users already familiar with UML**. It also consists only of rectangles, edges and text which facilitates manual drawing. *ShapeUML* uses a small number of visual variables and encodes a lot as text. For novice users it may be difficult to understand *ShapeUML* but optimizing it for novice users might introduce large deviations from UML which would make it harder for experts to understand.

ShapeVOWL *ShapeVOWL* uses a graph visualization with ellipses and edges. Experiments with other VOWL-based notations already suggest that **VOWL is intuitive** also for people with less knowledge about the underlying languages [4]. Additionally, semantic web experts are usually already familiar with different VOWL-based notations and the graph model in general; *ShapeVOWL* leverages this and **may provide a trade-off between understanding for experts and novices**. *ShapeVOWL* relies on simple geometric shapes and text, colors are optional, thus, with respect to perceptual discriminability, semantic transparency and visual expressiveness, *ShapeVOWL* can also be drawn by hand without effort (neglecting certain dual coding like more complicated icons).

4.9. Discussion

We analyzed both visual notations with respect to Moody's design principles and in the following discuss our findings which are summarized in Table 1.

One the one hand, *ShapeVOWL* uses more visual variables and symbols to express semantic constructs than *ShapeUML*. For example, it uses more *shapes*, meaning of *borders* but also *colors* and *icons*. This – in addition to the depiction of the underlying RDF graph data, specific edges to connect elements, and Venn diagrams – results in high scores for *semiotic clarity* and *semantic transparency*. All other principles are at least partially addressed with the exception of *complexity management* which can be accomplished by a tool implementing *ShapeVOWL*, e.g. by providing different means of zooming.

On the other hand, *ShapeUML* shows *semiotic clarity* and *graphic economy* with an advanced *cognitive fit*. This means that *ShapeUML* represents all RDF constraints' needed concepts in a cognitively manageable fashion and, additionally, may be suited for specific tasks and audiences. *Perceptual discriminability*, *semantic transparency* and *visual expressiveness* are affected by *cognitive fit* [8], thus, considering hand-drawn representations of *ShapeUML*, its simplicity may become an advantage as no special drawing abilities are needed.

Principle	ShapeUML	ShapeVOWL
Semiotic Clarity	+	++
Perceptual Discriminability	-	+
Semantic Transparency	-	++
Complexity Management	-	-
Visual expressiveness	-	+
Dual Coding	-	+
Graphic Economy	+	+
Cognitive Fit	++	+

Table 1

A comparative analysis with Moody's design principles [8] for cognitive effective visual notations reveals that ShapeVOWL scores better compared to ShapeUML. A double plus (++) indicates that each dimension of the principle is addressed, a single plus (+) that at least one dimension is addressed respectively not violated and a minus (-) indicates that a principle is not or very poorly addressed.

5. UnSHACLed editor

UnSHACLed is a graphical editor for RDF constraints. It allows users to validate RDF data against RDF constraints and view a validation report by loading existing RDF data into the tool and validate them with separately loaded or visually created RDF constraints. The main goal of *UnSHACLed* is to enable users familiar with RDF but not familiar with specific RDF constraint languages to create and edit RDF constraints. *UnSHACLed* offers a web interface and thus can be used with any browser. An early prototype was presented in previous work [9] and is available on GitHub[10]. In this paper we present a recently reworked version: https://github.com/KNowledgeOnWebScale/unshacled.

In this section we discuss features for an RDF constraint editor (Section 5.1) and how visual notations contribute to it, as well as introducing the implementation of our RDF editor *UnSHACLed* (Section 5.2).

5.1. Features for Data Shape Editing

In previous work [9] we introduced seven desired features for the editing of *data shapes*.

F1: Independence of constraint language Data shape editors should not confront domain experts with writing the textual syntax of a specific constraint language. Moreover, the visualization of the constraints should be independent of the underlying constraint language: generic (graphical) symbols can be used to (partially) hide language-specific textual syntax, as constraint languages have overlapping semantic constructs. Both *ShapeUML* and *ShapeVOWL* are constraint language independent and have a defined *visual vocabulary* covering semantic constructs of RDF constraints.

F2: Support multiple data sources Data shape editors should support domain experts in defining data shapes referring to multiple data sources at once. The proposed visual notations allow to define RDF constraints in a visual fashion for different data sources.

F3: Support different serializations Data shape editors should not restrict domain experts to specific serializations of the data source nor the constraint language. A data graph can be serialized in different ways without changing the actual data or structure (e.g. RDF/XML vs Turtle). The visual vocabulary of both *ShapeUML* and *ShapeVOWL* covers semantic constructs of RDF constraints and is currently mapped to SHACL. Thus it is represented in RDF which can be serialized to different serializations.

F4: Support multiple ontologies Data shape editors should support domain experts in defining data shapes for data graphs annotated with multiple ontologies simultaneously. Both notations use URIs where necessary, e.g. for property paths or class constraints. Thus, multiple ontologies are supported by both notations.

F5: Multiple alternative modeling approaches Data shape editors should enable and support multiple alternative modeling approaches and allow domain experts to choose the most adequate one for their needs. Two modeling approaches, complementary to visual notations, were discussed in our previous work [9].

F6: Non-linear workflows Data shape editors should allow domain experts to keep an overview of the relationship between the data graph and data shapes, by providing non-linear editing. Although the data graph is not visualized together with the shapes graph, the data is visualized in the data panel. Terms related to data shapes' assignment to instance data is covered by the visual notations, i.e. the *appliesOn* concept indicating on which data shown constraints apply by default.

F7: Independence of execution Data shape editors should allow importing and exporting the data shapes specified by the domain experts, as a use case may require to execute the data shapes elsewhere. Both *ShapeUML* and *ShapeVOWL* are currently mapped to

[10] https://github.com/dubious-developments/UnSHACLed

SHACL and, thus, to RDF which provides interoperability and allows the import and export of data shapes.

5.2. Implementation

We describe the modular architecture of our RDF constraint editor *UnSHACLed* (Section 5.2.1), and relevant GUI components providing user interactions in a visual fashion (Section 5.2.2).

5.2.1. Architecture

UnSHACLed is a web-based RDF constraint editor independent from specific data formats, visual notations or validation engines.

Framework *UnSHACLed* is implemented with the web framework emphVue.js following the *model-view-viewmodel* (MVVM) design pattern introduced by John Gossman in 2005[11].

It therefore can run in modern Browsers and no additional server infrastructure such as databases are required.

Intermediate format *UnSHACLed* uses the *state management pattern and library Vuex* to store RDF constraints using an intermediate data format which allows all application components to access the RDF constraints in a controlled manner. Therefore other constraint languages can be supported by providing a mapping to the intermediate format without the need to change other parts of the implementation.

Visual notations *UnSHACLed* uses the *VueKonva* library to draw canvas graphics. Several components for both *ShapeUML* and *ShapeVOWL* were developed to render the two notations. New visual notations can be added in the form of new components which also read and write data to the intermediate format of Vuex store.

Validation For validation the intermediate format is transformed to a representation of a concrete RDF constraint language (currently supported is SHACL) and is passed together with the data to a separate *validation engine*. Another constraint language and validation engine can be used which only leads to adjustments in *UnSHACLed* with respect to transformations of the intermediate representation or invocation of another validation engine, no adjustments to the GUI are required.

[11]https://web.archive.org/web/20051029151624/http://blogs.msdn.com:80/johngossman/archive/2005/10/08/478683.aspx

5.2.2. Graphical User Interface

In this section we discuss the graphical user interface of *UnSHACLed*, namely the different existing panels and interactions elements with which users can interact using visual notations.

Panels The GUI consists of three panels representing different parts of a Linked Data validation workflow: a data panel, modeling panel and validation result panel.

The **Data panel** shows data which should be constrained or described (left panel in Fig. 7). RDF is currently supported in different serializations, such as *turtle* and *JSON-LD*. This is raw data and can also be edited. *UnSHACLed* is modular and the functionality can be extended to also visualize data of other kind to support other *editing approaches*.

The **Modeling panel** shows RDF constraints in the visual notation chosen in the menu, both *ShapeUML* and *ShapeVOWL* are supported (middle panel in Fig. 7). Elements in the modeling panel are denoted visually and scalability is addressed with geometric zooming.

The **Validation result panel** shows the validation result of applying the *RDF constraints* of the *modeling panel* on the data of the *data panel* as reported by a *validation engine for RDF constraints*. The validation result panel is implemented as a modal dialog, i.e. it appears after clicking the *validation button*. *UnSHACLed* is independent of concrete *RDF constraint languages*, it can be extended with different validation engines.

Interactions Visual notations specify how RDF constraints are visualized, but *UnSHACLed* also allows to interact with the visualizations. Most notably nodes in the graph can be dragged and dropped inside the *modeling panel*. When hovering over an element a red and a green button appear representing actions for delete and editing. In the latter case a modal dialog opens in which users can change or add constraints. Thus, users can also edit RDF constraints using the visual notations and do not have to learn a specific textual syntax.

6. User Evaluation

We conducted a comparative study to validate our main hypothesis that *users familiar with Linked data can answer questions about visually represented RDF constraints more effective with ShapeVOWL than with ShapeUML*. We compared how accurately users can answer questions about data shapes represented using either *ShapeUML* or *ShapeVOWL*. In Section 6.1, we describe the questionnaire to cover various aspects of

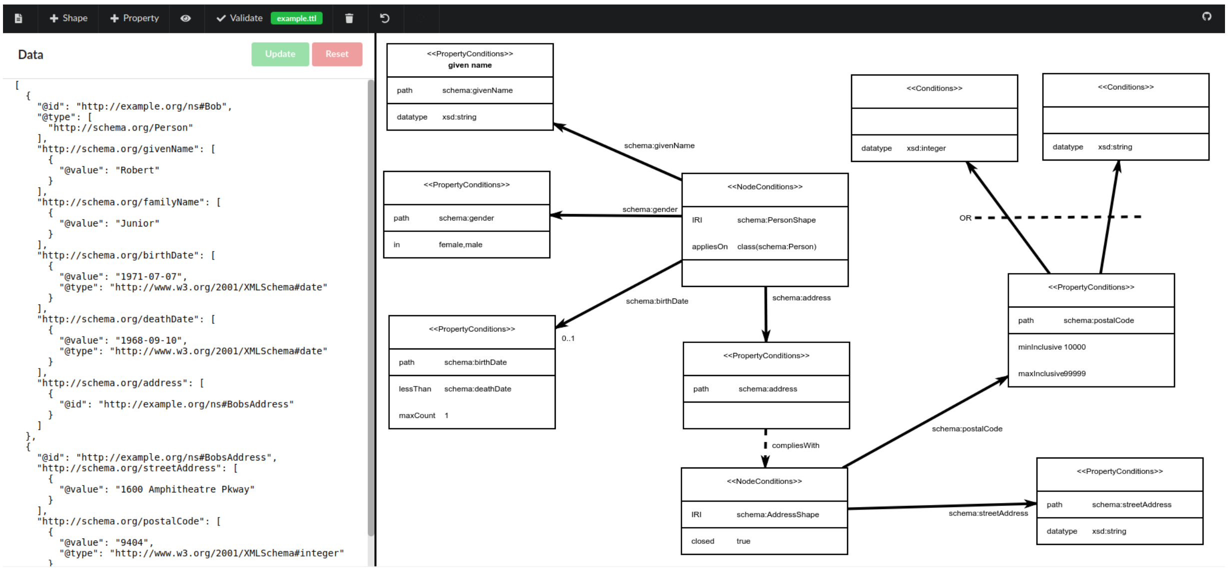

Fig. 7. The user interface of our tool UnSHACLed consisting of several panels supporting different editing approaches.

the data shape domain based on the SHACL core specification. In Section 6.2, we elaborate on the experiment, in Section 6.3, we discuss potential threats to validity, in Section 6.4, we analyze the results of quantitative questions, and in Section 6.5, we analyze results of qualitative questions. Collected (anonymized) data, the questionnaire and user introductions as well as code for the quantitative and qualitative analysis are openly available at https://doi.org/10.6084/m9.figshare.13614440.v1.

6.1. *Questionnaire*

We derived questions from the SHACL specification relevant to RDF constraints and validation, which were used in a user study to validate our hypothesis.

We created questions to test (i) at least one constraint type per core constraint category of the SHACL specification, and (ii) other RDF constraint concepts relevant for validation, i.e. the targeting mechanism, property paths, severity and deactivation. The SHACL specification lists eight core constraint categories:

1. value type, 1 constraint
2. cardinality, 1 constraint
3. value range, 1 constraint
4. string-based, 1 constraint
5. property pair, 1 constraint
6. logical, 1 constraint
7. shape-based, 2 constraints
8. *other*, 2 constraints

We selected at least one constraint type for each category and created an associated question, for example "*How many datatype constraints can you see?*" for the constraint type *datatype* of *value type* category. The last two categories mix several relevant constraint types, so, we selected 2 constraints types for each.

Additionally we created one question for each of the aforementioned other relevant concepts, such as "*How many property conditions with the severity 'information' can you see?*" for the concept *severity* or "*How many zero-or-more property paths can you see?*" for the concept *property paths*.

6.2. *Method*

The user study follows a *within-subject design* (also referred to as *within-group* or *repeated measures* [41]) in which all participants are confronted with examples of both visual notations *ShapeUML* and *ShapeVOWL*. We discuss the method of the user study by explaining the procedure, elaborating on recruited participants, and introduce the example test cases.

Procedure Potential participants with Linked Data knowledge were directly contacted by the authors. Those who participated were assigned in a round-robin fashion to one of two groups (groups A and B) to mitigate order effects (see threats to validity Section 6.3), and had to (i) read introductions to both *ShapeUML* and *ShapeVOWL* (presented in this order), and (ii) complete an online questionnaire. The user study is divided into three steps: a pre-assessment, a session in which the questionnaire is answered, and a post-assessment.

(i) The **pre-assessment** is focused on the participants' sociodemographic traits, such as year of birth, gender, and level of education, to provide indicators of the studied population. Additionally, through a self-assessment, the participants' expertise with Linked Data and with RDF constraints is assessed as well as their familiarity with the topic and tools.

(ii) The **main questionnaire** consists of 11 questions about data shapes presented using *ShapeUML* and *ShapeVOWL* to assess how effective visualized elements are recognized. After that, 15 questions on 4 test cases were asked to compare visualizations in *ShapeUML* and *ShapeVOWL*. These questions include 14 questions derived from the SHACL specification (Section 6.1) and one open question to provide feedback about the shown examples and asked questions. For group A the general example is first shown in *ShapeUML* afterwards in *ShapeVOWL* and then the test cases are presented started with the first test case in *ShapeUML*, the second in *ShapeVOWL* and so forth; it is the other way around for group B to mitigate order effects (see validity threats in Section 6.3).

(iii) The **post-assessment** consists of 4 questions and collects information about the participants' preference for either ShapeUML or ShapeVOWL to answer questions about data shapes, whether they want to use one of the notations also for the editing of data shapes, besides only to visualize them; and general feedback.

Participants The online questionnaire was sent to 14 potential participants in September 2020. 12 participants took part in the experiment, their age range was 23 to 40. All participants were highly educated: all have at least a master degree, one a PhD. According to a self assessment, all participants are familiar with Linked Data, most participants generate or use Linked Data (Fig. 8). All participants are familiar with *UML class diagrams*, the underlying notation of *ShapeUML*, and the majority of the participants is familiar with the tool *WebVOWL*, a tool implementing *VOWL*, the underlying notation of *ShapeVOWL* (Fig. 9).

Real world test cases All test cases are real world from online resources such as GitHub or the *ShapeViBe* benchmark.

The **Traffic Lights** test case represents constraints on RDF lists[12]. It is characterized by containing a *zero-or-more property path* and several constraints on RDF list elements while also reusing an external data shape by referring to it with a constraint.

The **Address** test case is an excerpt from possible schema.org data shapes[13]. It was manually curated to constrain schema.org addresses for Australia. This test case is characterized by containing logical constraints as well as a few other constraints on literal values.

The **DCAT** test case is an excerpt from the DCAT application profile for Swiss data portals[14]. It has constraints on many properties of a single node, mostly constrained by their cardinality, datatype or class, but also by logical constraints, e.g. either class A *or* B.

The **Geo coordinates** test case is from the *ShabeViBe* benchmark[15]. It is characterized by containing combinations of different minimum and maximum constraints which can be easily confused. Namely, min/-max cardinality constraints on properties, min/max value range constraints on property values as well as qualified cardinalities related to data shapes.

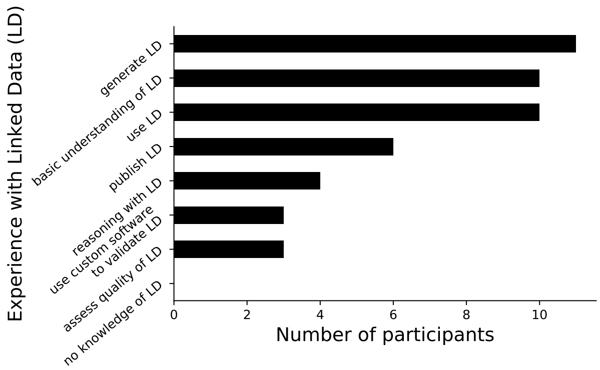

Fig. 8. Answers based on self assessment: all participants are familiar with Linked Data, most participants generate or use Linked Data.

6.3. Threats to Validity

External and internal threats to the experiment's validity exist, we identified the following threats and we discuss how we addressed them in our study design.

6.3.1. External Validity Threats

External validity threats occur when wrong inferences from sample data are made beyond the stud-

[12] https://www.topquadrant.com/constraints-on-rdflists-using-shacl/

[13] http://datashapes.org/schemashacl.shapes.ttl

[14] https://github.com/factsmission/dcat-ap-ch-shacl

[15] https://w3id.org/imec/unshacled/shape-vibe/modules/min-max-values/

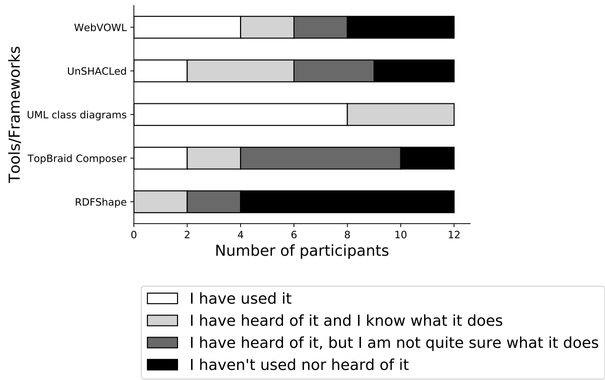

Fig. 9. UML diagrams known by all participants and already used by the majority, other tools/frameworks less commonly known.

ied population or experimental setup [41]. We identified two external threats: **participants familiarity with Linked Data** and **experiment environment**.

Participants familiarity with Linked Data This threat concerns the generalization to individuals outside the study [41]. All our participants were recruited from Ghent University, Belgium and RWTH Aachen, Germany and were familiar with Linked Data, thus the findings might not be generalizable to a more general population. However, this was intentional as we aimed to study users already familiar with RDF graphs, a prerequisite to understand RDF constraints which are the semantic constructs our visual notations represent.

Experiment environment This threat concerns the generalization to individuals outside the experiment's setting [41]. The experiment was an online questionnaire. Participants could use any browser and computer, thus, they participates from a well-known environment. No specific experimental setup prevents generalizations to individuals outside our study.

6.3.2. Internal Validity Threats

Internal validity threats concern the experimental setup or experience of participants which threaten the ability to draw correct conclusions about the population in the experiment [41]. We identified three internal threats: **selection bias**, **sample size** and **order effects**.

Selection bias This threat concerns the selection of biased participants, i.e. participants with certain characteristics that predispose them to have certain outcomes [41]. Our participants were all recruited from Ghent University and RWTH Aachen and have similar demographics. All participants have knowledge about Linked Data, but this is intentional as it is a prerequisite of the user study. To mitigate a selection bias all participants were assigned in a round-robin fashion to one of two groups, i.e. groups were not assigned based on specific characteristics. Some participants might be more familiar with one of the underlying visual notations of *ShapeUML* or *ShapeVOWL*. However, they self-assessed their familiarity with *UML class diagrams* and the *WebVOWL* tool in the pre-questionnaire, therefore any bias is visible. Please note that familiarity with one of the notations is considered positive as the design rationale of both visual notations is to build upon the underlying visual notation.

Sample size A small sample size may not have sufficient statistical power to detect an effect. Our sample size is relatively small. To mitigate this threat, we chose a *within-subject study design* [41]. It reduces errors associated with individual differences without requiring a large pool of participants[16].

Order effects When participants perform tasks several times certain effects like learning can occur. To counterbalance potential order effects when presenting *ShapeUML* and *ShapeVOWL*, we assigned participants in a round-robin fashion to two different groups. The first group (group A) started with the first example in *ShapeUML*, the second in *ShapeVOWL*, the third in *ShapeUML* and so forth. Participants of the second group (group B) were presented the first example in *ShapeVOWL*, the second in *ShapeUML* and so forth.

6.4. Quantitative Results

We analyze the participants' self assessment given by a Likert scale [42] (Section 6.4.1), statistically validate the significance of the overall error rate differences between *ShapeUML* and *ShapeVOWL* (Section 6.4.2), analyze error rates per RDF constraint concept (Section 6.4.3), and analyze error rates per real world test case (Section 6.4.4).

6.4.1. Self Assessment

The post-questionnaire contained three questions in which the participants could self assess how *confident* they are with their answers, if they prefer *ShapeVOWL* over *ShapeUML* and if they would like to use *ShapeVOWL* also for RDF constraint editing. These three questions were asked using a 7-point Likert scale from 1 (not agree at all) to 7 (fully agree).

[16]https://web.archive.org/web/20201216150003/http://onlinestatbook.com/2/research_design/designs.html

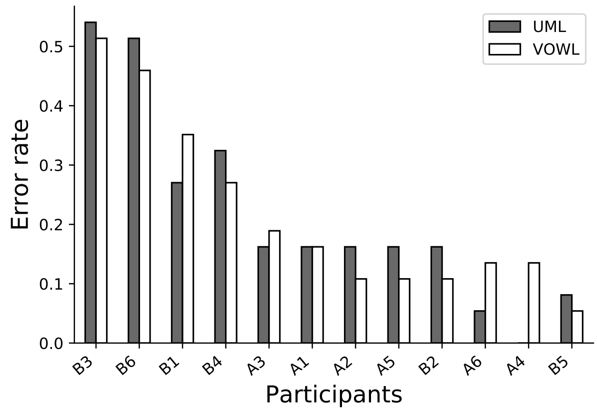

Fig. 10. Error rates of participants: 7 participants made fewer errors using *ShapeVOWL*; participants of group B have higher error rates in general. According to a Wilcoxon signed-rank test the mean error differences are not statistically significant: p-value$=0.8933$

All participants were asked if they are confident that their provided answers are correct. Their average value is 3.6 and median is 3, thus in a self assessment **participants are not very confident**. Participants could also provide feedback for each test case via a text field. Considering the provided feedback, some participants had trouble interpreting the asked questions which could relate with their low confidence.

All participants were asked if ShapeVOWL is preferred and the average value is 4.6 and median is 5, thus in a **self assessment participants prefer ShapeVOWL**. Similarly the average is 4.8 and median is 5 for the question if the participants would like to use ShapeVOWL to edit RDF constraints.

6.4.2. *ShapeUML/ShapeVOWL Error Rate*

Based on the correct answers, we calculated the error rates of all questions to compare *ShapeUML* and *ShapeVOWL*: initial questions for general examples as well as for the 4 test cases (Section 6.1).

Statistical test **There is no significant difference in the mean error rates of *ShapeUML* and *ShapeVOWL***. The error rate for *ShapeVOWL* is lower for 7 from 12 participants compared to *ShapeUML*. However, we are interested if the mean error difference is statistically significant. We first tested the normality of the error rates' distribution using a distribution plot and a *Shapiro-Wilk test* [43] with $\alpha = 5\%$ to determine which statistical test to choose. The data was not normally distributed, thus we performed a *Wilcoxon signed-rank test* [44] with $\alpha = 5\%$. The calculated p-value of 0.8933 is bigger than α so we fail to reject the null hypothesis, which means there is no significant difference in the mean error rates.

6.4.3. *Error Rates for Constraint Concepts*

The questions of Section 6.1 represent fundamental concepts and core constraints of RDF constraint languages. We analyze which concepts were processed most or least effective, by comparing the error rates of questions across test cases (Fig. 11).

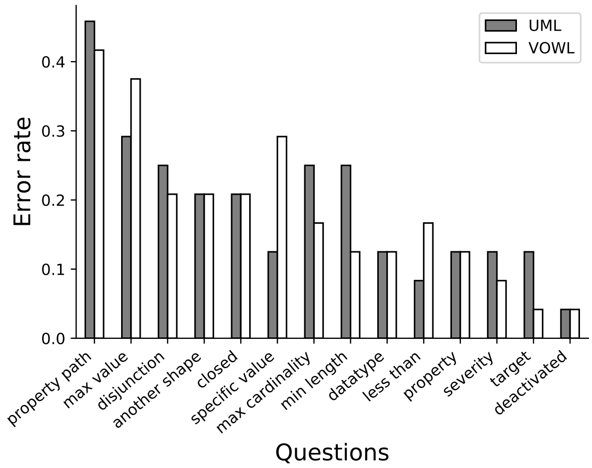

Fig. 11. Errors were made with both visual notations. Higher error rates for both visual notations were achieved for *property paths* and *maximum value*.

With both visual notations the error rates are relatively low, on average 19% for *ShapeUML* 18% for *ShapeVOWL*. We discuss each question relating to one RDF constraint concept.

Deactivation of data shapes **Both ShapeUML and ShapeVOWL make it easy to recognize deactivated data shapes**. This concept indicates that *data shapes* are not considered for validation. With both notations only 4% errors were made, one reason might be that this concept is also visually represented in both notations. It is indicated by striked through text in *ShapeUML* and by dashed borders in *ShapeVOWL*.

Target concept **Participants recognize the target concept more effective with ShapeVOWL**. This concept indicates on which nodes of the data graph, *data shapes* apply by default. With *ShapeVOWL* only 4% errors are made compared to 12.5% errors with *ShapeUML*. Both notations use the label "appliesOn": *ShapeUML* lists it in the middle compartment and *ShapeVOWL* in a note-element attached to the note. Since both notations encode *targeting* textually at *node shape*, the error difference occurs due to other reasons, e.g. general participants performance.

Severity of constraints **It is easier to spot severities with ShapeVOWL than with ShapeUML**. This concept indicates a severity which after a validation may be indicated in the validation report, possible values are "violation" (the default), "warning" and "information". Fewer errors were made with *ShapeVOWL* which indicates severities additional to text with border colors. With *ShapeUML* severities are only encoded with text which suggests that the *dual coding* design principle (Section 4) improves *ShapeVOWL*.

Property constraint **Both notations facilitate the recognition of property constraints relying on the RDF graph model's visualization**. This concept links contextualized constraints for nodes (node shape) with contextualized constraints for properties (property shapes). With *ShapeUML* and *ShapeVOWL* 12.5% errors were made, both notations encode this constraint type similarly which may explain the similar error rate: both notations visually represent *node and property shapes* with geometric shapes connected with arrows labeled with the property.

Less than or equals constraint **Effective processing with ShapeVOWL may rely on text as well**. This concept represents that one property value must be less than the value of another property. Participants made 8% errors with *ShapeUML* and 17% with *ShapeVOWL*. Both notations encode this constraint type as text, *ShapeVOWL* additionally uses an icon but in the provided example the text "lessThanOrEquals" was omitted for *ShapeVOWL*. This is interesting as it shows that an icon alone may not provide sufficient information and text in the *dual coded* principle is indeed necessary for RDF constraints too.

Datatype constraint **ShapeUMLs clear textual representation of datatype constraints was recognized as accurate as ShapeVOWL's visually enhanced representation**. This concept represents the constraint that a literal value must be of a certain datatype. With both notations 12.5% of answers were wrongly answered. Whereas *ShapeUML* represents this constraint as text only, *ShapeVOWL* uses an additional icon and relies on the VOWL notation representing literal values as yellow rectangles. Despite all the visual features but maybe because of some, the error rate for *ShapeVOWL* is not lower: we represent literal values like VOWL with yellow rectangles which might be counted already as datatype constraint by some participants possessing prior knowledge of VOWL.

Minimum length constraint **Visual features of ShapeVOWL such as position may improve ShapeVOWL's effectiveness for minimum length constraints compared to ShapeUML**. This concept represents the constraint that a property value must be of a certain minimum length, i.e. minimum string-length for literals and IRIs. Double the number of errors were made with *ShapeUML*, 25%, compared to *ShapeVOWL*, 12.5%. Whereas *ShapeUML* clearly indicates "minLength", *ShapeVOWL* uses the notation "length(min..max)" positioned next to the literal and uses an additional icon. A combination of visual features or one of it may cause lower error rate for *ShapeVOWL*, i.e. combination of *position*, *label* or *icon*.

Maximum cardinality constraint **Participants make less mistakes in recognizing maximum cardinality constraints with ShapeVOWL**. This concept represents the constraint that a property must have a maximum cardinality. Participants only made 17% errors with *ShapeVOWL* compared to 25% errors with *ShapeUML*. Both notations indicate the cardinality next to the arrow head connecting node with property shapes. One reason for the higher error rate of *ShapeUML* might be that other constraint types starting with "max" such as "maximum value" might have been mistakingly counted. *ShapeVOWL* also uses the visual variable *position* which distinguishes property cardinalities (next to the arrow head) from other minimum/maximum constraint types shown in note-elements such as *maximum length*.

Specific value constraint **Participants recognized constraints restricting property values to explicitly provided valid values better with *ShapeUML***. The question related to this concept targeted constraints which limit the value of a property to one specific value which is provided directly or provided in a list of valid values. We observe almost double the number of errors for *ShapeVOWL*, 29%, compared to *ShapeUML*, 12.5%. Both notations use the same labels for these constraints, i.e. "hasValue" for a single value and "valueIn" if a list of valid values is given. *ShapeUML* regularly lists these constraints in the lower compartments of *property shapes* and *ShapeVOWL* lists them in a note-element next to the literal of the property shape they apply on. It is possible that some participants did not count the case in which a whole list of valid values is provided, however this does not explain the comparable higher error rates for *ShapeVOWL*. One participant pointed out that a question asking for a "specific value" might comprise other constraint types too, which suggests a too generic question phrasing.

Comply with constraint **We did not observe error differences between ShapeUML and ShapeVOWL for the similarly represented comply with constraint**. This concept represents the constraint that (a specific number of) property values must comply with a *data shape*. With *ShapeUML* and *ShapeVOWL* 21% errors were made. Both notations encode this concept similar, a dashed directed edge from one *data shape* to another with the label "complyWith" and optional qualified cardinalities at the source of the edge, and, thus in a visual fashion. Participants sometimes identified this constraint type even when it was not present in a test case which indicates that also other constraints were identified as "compliant with" suggesting a too generic "comply with" label.

Closed data shapes **Participants may misunderstood underlying semantic constructs**. This concept represents the constraint that a node in the data graph must only have properties for which property constraints are specified, i.e. the node is closed in the sense that no other properties are allowed. With both notations 21% errors were made. Whereas *ShapeUML* only encodes this constraint textually, *ShapeVOWL* additionally indicates "closeness" using a thick border; both notations use the label "onlyListedProperties". Several constraints of this type were identified in test cases where it was not present and in a few cases it was not identified when it was present. Since errors were also made with the textual only *ShapeUML* representation we can exclude visual features as possible misunderstanding and it may indicate that participants did not understand the concept behind this constraint type, i.e. the semantic construct and not its visual representation.

Disjunction constraint **ShapeVOWL disjunctions were slightly more often correctly identified compared to ShapeUML**. This concept represents logical disjunction constraints. Participants made more errors with *ShapeUML*, 25%, than with *ShapeVOWL*, 21%. Both notations represent this constraint type visually, *ShapeUML* relies on specific edges following the *UML* notation and the label "OR" and *ShapeVOWL* uses a dedicated node with label "OR" and icon representing a Venn diagram[17]. The additional icon of *ShapeVOWL* may have caused the slightly better scores as it "pops out", but with both notations a variety of different answers were provided. It seems that participants counted the number of *data shapes* connected with a disjunction or their cardinalities rather than count the occurrence of a single disjunction.

Maximum value constraint **Participants spotted maximum value constraints better with ShapeUML**. This concept represents that a literal value must not exceed a maximum value. Participants made fewer errors with *ShapeUML*, 29%, compared to *ShapeVOWL*, 37.5%. *ShapeUML* represents this constraint type using the label "maxExclusive" or "maxInclusive", whereas *ShapeVOWL* uses the single notation "range(min..max)" visualized in a constraint note-element next to the literal the constraint applies on, and, thus, also the visual variable *position*.

The observations are interesting as we expected fewer errors with *ShapeVOWL* due to its better cognitive features. One participant, according to provided feedback, did not understand the difference between the questions for *maximum cardinality* and *maximum value*. On the one hand this could also explain the error rates for other participants. On the other hand the visual features of *ShapeVOWL* were designed to avoid such an issue (using position and different labels).

Property Paths **Most participants successfully recognized property paths but some may confused them with logical relationships**. This concept is used to define reachable objects from subjects, i.e. to define on which reachable properties constraints apply. This concept resulted in the highest error rates, but slightly fewer errors were made with *ShapeVOWL*, 42%, compared to *ShapeUML*, 46%. Both notations express a property path as atomic value as label of a relationship connecting *node* with *property shapes*. More than 50% of participants successfully recognized this concepts using its textual representation. However, the provided answers suggest that participants may have confused property paths with a combination of logical relationships with cardinalities on properties.

6.4.4. Error Rate for Test Cases

The participants saw an initial example in both *ShapeUML* and *ShapeVOWL* and then received four real world test cases. We elaborate on error rates of different test cases, as described in the procedure section.

Constraint type distribution and general group performance need to be considered when interpreting the results. As shown in previous work, certain constraint

[17] In the user study test cases the Venn diagram was slightly different from the ShapeVOWL specification

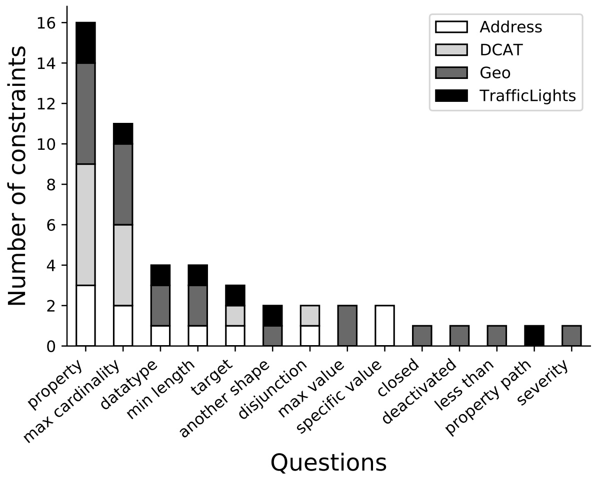

Fig. 12. The occurrence of question-related constraint types in the four real-world test cases. Two constraint types were present in all examples, seven constraint types only in one example.

types are used more often than others [12], resulting also in unequally distributed constraint types in our real world test cases (Fig. 12). Additionally, due to our study design, group B participants have seen the test cases *Address* and *Geo coordinates* in *ShapeUML* and the *Traffic Lights* and *DCAT* in *ShapeVOWL* (Fig. 13).

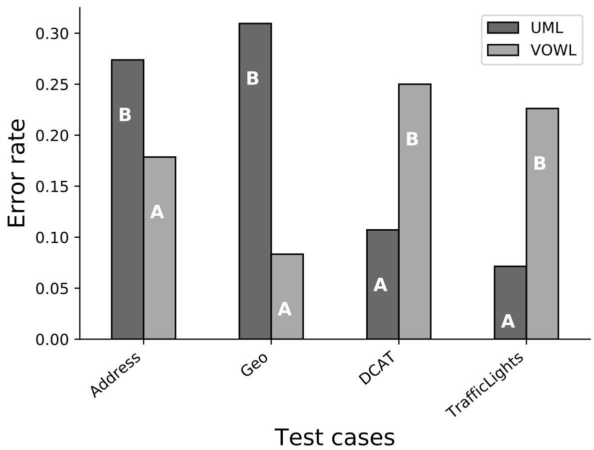

Fig. 13. Participants of group B have higher error rates for both notations. Each test case was only seen by one group in one notation, suggesting that a bad performance for one visual notation is not only related to the notation but also the participant group.

In half of the test cases fewer errors were made with *ShapeUML* and in the other half fewer errors were made with *ShapeVOWL*. Traffic light example: fewer errors with *ShapeUML* (7% vs 23%). Australian address example: fewer errors with *ShapeVOWL* (18% vs 27%). DCAT example: fewer errors with *ShapeUML* (11% vs 25%). Geo coordinates example: fewer errors with *ShapeVOWL* (8% vs 31%). We elaborate for the general example and the different real world test cases to analyze what constitutes these error values.

Error Rate for General Examples Initially, participants are presented a general example to test their understanding, it is the first example they see after reading the introductions of both visual notations. **With both notations the concepts *closed*, *datatype* and *target* were processed with more than 80% correctness**. Interestingly there were no wrong answers for the concept *closed* with *ShapeUML* whereas the – according to the theory – more cognitive effective *ShapeVOWL* lead to 16% wrong answers. Both notations use the same textual label and *ShapeVOWL* additionally encodes this concept visually.

More than 50% errors were made with the concepts *severity* and *minimum and maximum cardinality* in both notations. **Compared to *ShapeUML*, *ShapeVOWL* resulted in 30% fewer errors for the concept *severity*** most likely because it dual codes severities, i.e. encode them textually and additionally visually. In accordance with the user introduction, the default severity "violation" – which was asked for – was not indicated textually because it is the default. However, with *ShapeVOWL* the concept *severity* is dual coded and colored borders were still present.

Participants identified too few *cardinalities* with both notations, the frequently given wrong answer of 5 indicates that participants only counted pairs of minimum and maximum only once, ignoring that according to the question they should *not* count zero and infinity cardinalities and hence count minimum and maximum separately. Two participants identified no cardinalities at all suggesting it was not clear what cardinalities are or the answer was given by accident.

Address This test case resulted in similar error rates for both *ShapeUML* and *ShapeVOWL*.

Questions related to the concepts *target*, *property* and *deactivated* were answered correctly with both notations by all participants. Errors were made with *ShapeUML* for the concepts *severity*, *less than*, *max cardinality*, *disjunction* and *comply with* **whereas with *ShapeVOWL* no errors were observed for these concepts**. Two out of these five concepts are similarly encoded in both notations (max cardinality and comply with), but the three concepts *severity*, *less than* and *disjunction* have more visual features which may explain that compared to *ShapeUML* no errors were made.

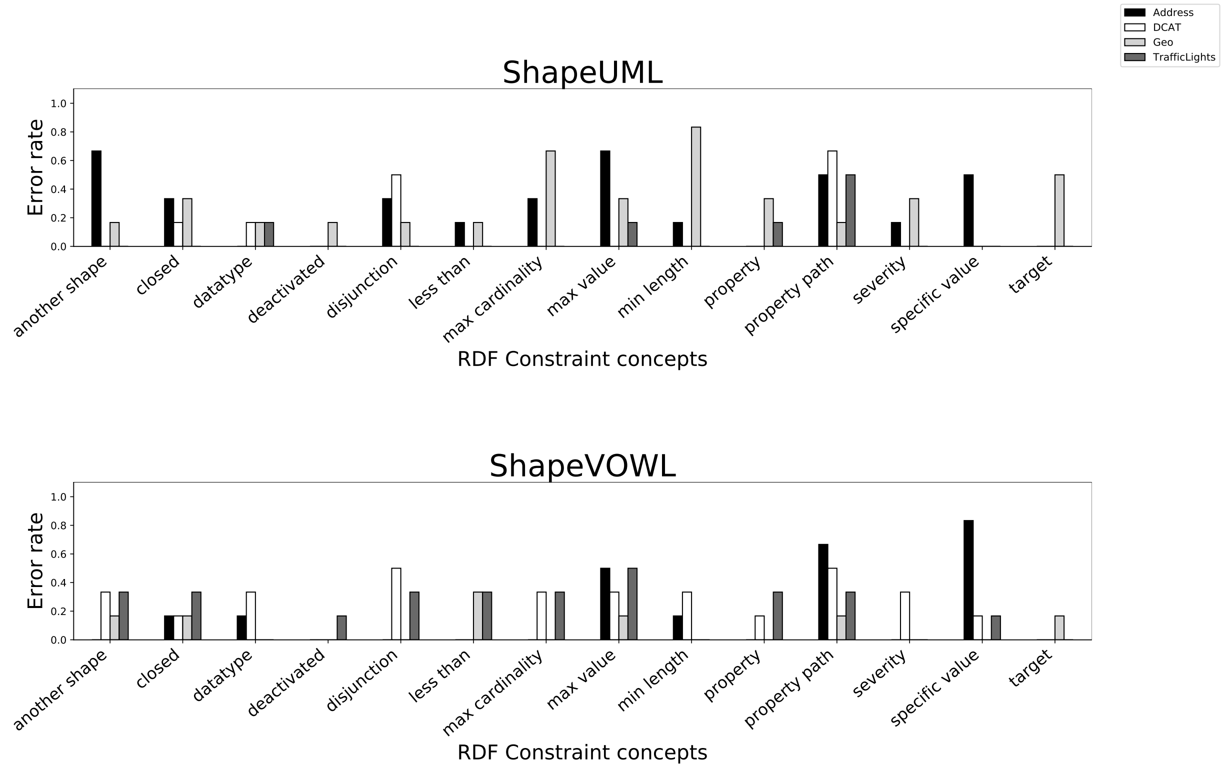

Fig. 14. The error rates for the different questions across the 4 real world test cases. Most RDF constraint concepts related questions were answered correctly. Participants made the most errors for *property paths*, *maximum value*, *specific value* and *disjunction*.

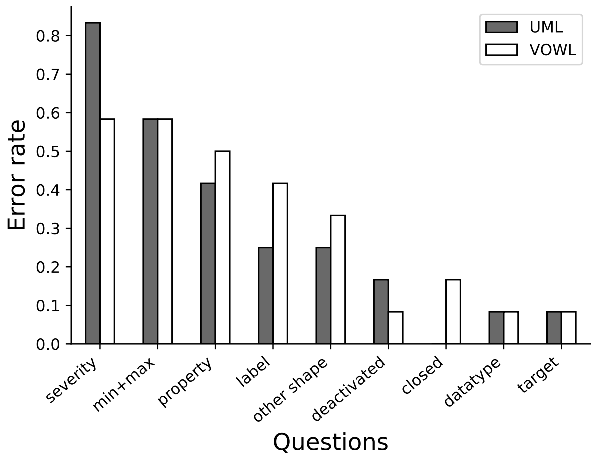

Fig. 15. The error rates of the general example which participants saw first after reading the introductions. More than 80% correct answers were given for the concepts closed, datatype and target whereas for severity relatively high error rates were achieved.

For the concepts *datatype*, *property path* and *specific values*, *ShapeVOWL* performs worse compared to *ShapeUML* in this test case. Some participants identified only one *specific value* constraint instead of two. This suggests that either the *hasValue* or the *valueIn* constraint was counted, instead of both. One participant pointed out in the feedback question that due to a shown *negation* constraint only a *not specific value* is present which resulted in the interpretation that 0 *specific value* constraints exist in this test case. **These observations regarding different interpretations hint that some constraint types conceptually allow several interpretations**.

Geo coordinates example This test case from the *ShapeViBe* benchmark resulted in the **highest error rate for *ShapeUML*** with 31%, and in the lowest error rate for *ShapeVOWL* with 8%. Both visual notations use different terms to distinguish these different *minimum* and *maximum* constraint types (Figs. 1 and 5).

One reason for the low error rates could be that *ShapeVOWL* uses **different terminology and icons to distinguish the different *minimum* and *maximum* values which makes them easier distinguishable**; for *ShapeUML* errors were made for all min/max constraints, whereas for *ShapeVOWL* there are no errors for *min length* and *max cardinality*.

DCAT This test case which is an excerpt from the DCAT application profile for Swiss data portals resulted in the the highest error rate for *ShapeVOWL*.

All participants answered more than 70% of questions correctly with *ShapeUML* and made only minor errors for *datatype* and *closed* constraints as well as some more errors with *property paths* and *disjunction*. Participants identified *datatype* constraints although non were present, similarly they identified non-existent *closed* constraints. One *property shape* contained a *nodeKind sh:Literal* constraint and thus participants may have confused this with a datatype. The example also contains a *class constraint* with the value `schema:URL` which officially is a datatype, users familiar with this term may have counted it as datatype; this real world test case uses `schema:URL` wrongly as class, however this example explicitly marks it with a *class constraint*.

Participants scored relatively **high error rates for the *zero-or-more property path* and *disjunction*** constraints with both notations. No such property path was present in the example, but an existing exclusive disjunction constraint with the label "OneOf" may have been confused by some participants with *property paths*). Additionally, some participants wrongly counted the exclusive disjunction with the label "OneOf" when counting disjunctions and asked for "disjunction (logical or)".

Traffic Lights This test case representing constraints on RDF lists resulted in the the **lowest error rate for *ShapeUML*** and is the only test case actually containing explicit *property paths*.

Similarly to the *DCAT test case*, more than 70% of questions were correctly answered by all participants using *ShapeUML*; considering both notations questions related to the concepts *target*, *severity* and *minimum length* were answered correctly by all participants.

Error rates of 50% occurred for *property paths* in *ShapeUML* and *maximum value* for *ShapeVOWL*. Based on the provided answers it seems that some participants used the *minimum* or *maximum* cardinality of the *property paths* value instead of counting once the only existing *zero-or-more property path*.

6.5. Qualitative analysis

Qualitative feedback is derived from each test case in the main questionnaire and generally for both notations in the post assessment. We qualitatively analyze provided answers for the post assessment following a common data analysis for qualitative data [41]: we explain the used analysis method, and present the results. In total 58% of participants answered this question.

6.5.1. Method

A general procedure for a qualitative analysis involves the process of "coding" [41], a commonly used technique for reducing qualitative data to meaningful information by assigning labels to chunks of data [45]. Following common guidelines [41] we read answers provided in the post questionnaire and thus were able to identify 5 high level codes: advantages, disadvantages, uncertainty, suggestion and preference. These codes are further detailed in a hierarchy, for example the high level code *advantages* is further specified as *easier comprehensible*, *display of sparse constraints* and *space efficiency*. In a similar fashion the other high level codes are further specified to be used as annotation for the qualitative data.

6.5.2. Interpretation and Meaning

Based on the created annotations we interpret the feedback provided by the participants by discussing the high level codes such as *advantages* and which information specifically was provided.

Participants preference and suggestions **Findings regarding preferences and provided suggestions correspond with our analysis of cognitive effective design principles in both notations**, i.e. *ShapeVOWL* adheres to more design principles. In total 4 participants explicitly indicated which visual notation they prefer, 3 of them prefer *ShapeVOWL* and 1 *ShapeUML*. To improve *ShapeUML* one participant suggested to remove potential redundancies in *ShapeUML* (see disadvantages) and “a more user-friendly visualisation of UML (eg: colors, option to hide parts)”.

Advantages **Slightly more advantages were pointed out for *ShapeVOWL*, whereas both notations have their own advantages mostly related to how comprehensible they are for certain use cases**. In total 5 participants provided feedback with respect to advantages, 3 of them for *ShapeUML* and 4 for *ShapeVOWL*. *ShapeUML* was recognized more space efficient by 1 participant whereas the same participant mentioned that for sparse constraints *ShapeVOWL* “looks cleaner”. For both notations 3 participants indicated that the respective notation is easier comprehensible. For *ShapeUML* 2 participants pointed out that its list representation allows to condense more constraints of a single node and 1 participant expressed that *ShapeUML* is more intuitive. For *ShapeVOWL* 2 participants pointed out that it is easier to spot constraints due to visual features and 1 participant explicitly mentioned the familiarity to *VOWL* as reason.

Disadvantages **Although *ShapeVOWL* was preferred, more disadvantages were explicitly pointed out for it compared to *ShapeUML*.** In total 3 participants provided feedback with respect to disadvantages, 1 for *ShapeUML* and 3 for *ShapeVOWL*. *ShapeUML* was perceived redundant by 1 participant in a negative sense, i.e. the repetition of property paths both in the *data shape* rectangle and on the relationship between *node and property shape*. For *ShapeVOWL*, 2 participants reported possible complications when interacting with it, namely "many small comment boxes" for constraints which "would be less orderly" and the different geometrical shapes and colors as "things" which are "more of a hassle to work with (more clicking and less typing involved)". Additionally, 1 participant noted that *ShapeVOWL* "looks very simplistic, but needs more understanding to apply".

Uncertainty **Corresponding with Likert-scale answers regarding confidence and our quantitative analysis, participants explicitly mentioned unclear terminology**. In total 3 participants provided feedback with respect to unclarity, whereas 2 participants mentioned an unclear terminology and 1 participant ambiguous questions. This corresponds also with observations from the quantitative analysis, i.e. wrong answers for conceptually similar constraint types.

7. Discussion and Conclusion

Data integration as main challenge in our time can be addressed with the uniform graph data model of RDF. Use case specific data quality requires validation, but currently human users – often the creators of constraints – are not well supported when viewing and editing RDF constraints. Therefore, we investigated visual notations for RDF constraints tailored for the human information processing system to answer the research question *how we can support users in viewing and editing RDF constraints?*. The human information processing system requires effective visual notations that move the cognitive load from the slow cognitive processing to the fast perceptual processing.

The two visual notations *UML* and *VOWL* are broadly used within the Semantic Web community. We reused these already familiar to users notations and adapted them for RDF constraints: the two notations are dubbed *ShapeUML* and *ShapeVOWL*.

In particular, we investigated in this work the hypothesis that "*users familiar with Linked Data can answer questions about visually represented RDF constraints more effective with ShapeVOWL than with ShapeUML*", because *VOWL* was built with the aim to be intuitive. We could not validate this hypothesis: there was no significant difference in error mean values which would indicate that better results are achieved with *ShapeVOWL*. However, analyzing the design considerations of both visual notations and user study results in detail we conclude the following things.

ShapeVOWL is preferred Although our hypothesis regarding effective processing could not be validated in the performed user evaluation, **detailed findings of our work strongly suggest that *ShapeVOWL* will find more user acceptance than *ShapeUML*.** For both notations on average 81% of RDF constraint related questions where answered correctly. Several factors imply a higher user acceptance for *ShapeVOWL*: (i) according to our analysis based on cognitive effective design principles, *ShapeVOWL* adheres to more principles compared to *ShapeUML*, (ii) findings of our quantitative analysis suggest that visual variables such as position, border or icons positively influence *ShapeVOWL*s effectiveness in answering RDF constraint questions, (iii,) a self-assessment by users of our study revealed that *ShapeVOWL* is preferred, and (iv) according to our qualitative analysis users prefer *ShapeVOWL* as constraints can be spotted easier.

Disadvantages brought up in the qualitative analysis – such as *complicated interaction* or *space efficiency* – mainly concern more complex and dense RDF constraint graphs and can be mitigated by complementary functionality of RDF constraint editors implementing *ShapeVOWL*. Additionally, findings of our study with respect to misunderstood terminology or concepts (for both notations) can be addressed in future versions for both notations, e.g. more specific labels.

Clear and efficient text encoding of ShapeUML with potential improvement Despite visual features for cognitive effective processing by humans, we noticed that *ShapeUML*s textual representation in certain cases was as effective as *ShapeVOWL* and sometimes even more effective. According to our qualitative analysis, *ShapeUML* has an advantage for more dense or complex RDF constraint graphs due to its space efficient representation. Although text is processed using the slower *cognitive processing system* [8], this system might be needed for RDF constraints in any case. **But instead of providing an enhanced alternative notation such as *ShapeVOWL*, the already space efficient *ShapeUML* can be improved by addressing specific design principles to support users even better**. However, this would cause that *ShapeUML* may deviate from the *UML* specification, but as one participant put it: "I do believe a more UML-like format would be preferred by users IF [sic] users were allowed some slack from the rigid UML definitions".

Limitations Our work covers the accurate processing of visually represented RDF constraint concepts and, thus, does not cover scalability of visual notations or the speed in which users processed presented information. To the best of our knowledge this is the first work investigating visual notations for RDF constraints in detail. Hence our results are initial results. We studied how different RDF constraint concepts can be visualized and how this affects the accuracy of user-provided answers based on related questions.

Future Work Findings of our analysis suggest **future work regarding the integration of visual notations in RDF editors, the visual notations itself and, additionally, a possible mapping from ShEx concepts**. In future work we plan to incorporate features in our tool *UnSHACLed* to complement both visual notations such as semantic zooming to improve working with large RDF constraint graphs or enhanced user interactions to accommodate for different use cases; generally, more research towards user interactions is needed to understand real needs, especially with respect to different editing approaches for RDF constraints.

Regarding the visual notations, a visually enhanced *ShapeUML* variant – as suggested by a participant – could represent a trade-off in space efficiency and effective processing and it would be an appropriate candidate for future developments and user evaluations.

Finally, a mapping from ShEx concepts to the presented visual notations could motivate efforts to extend the presented tool *UnSHACLed* with respect to ShEx validation of RDF data, thus more users would profit from the developed effective visual notations.

Acknowledgements

We would like to thank the organizers and participants of the Open Summer of Code 2019 in Belgium in which an updated version of UnSHACLed was implemented. We also thank all participants of the user study for their time and efforts. The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (VLAIO), and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders (FWO).

References

[1] J.E. Labra Gayo, E. Prud'hommeaux, I. Boneva and D. Kontokostas, *Validating RDF Data*, Vol. 7, Morgan & Claypool Publishers LLC, 2017, pp. 1–328.

[2] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), Recommendation, World Wide Web Consortium (W3C), 2017. https://www.w3.org/TR/shacl/.

[3] J.M. Juran, *Juran's Quality Control Handbook*, 4th edn, Mcgraw-Hill, Texas, USA, 1988.

[4] S. Lohmann, S. Negru, F. Haag and T. Ertl, Visualizing ontologies with VOWL **7** (2016), 399–419.

[5] P. Heyvaert, A. Dimou, B. De Meester, T. Seymoens, A.-L. Herregodts, R. Verborgh, D. Schuurman and E. Mannens, 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** (2018), 31–50.

[6] F. Haag, S. Lohmann, S. Siek and T. Ertl, QueryVOWL: A Visual Query Notation for Linked Data, in: *Proceedings of ESWC 2015 Satellite Events*, Vol. 9341, 2015, pp. 387–402.

[7] M. Weise, S. Lohmann and F. Haag, Extraction and visualization of tbox information from sparql endpoints, in: *European Knowledge Acquisition Workshop*, 2016, pp. 713–728.

[8] D. Moody, The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering, *IEEE Transactions on Software Engineering* **35**(6) (2009), 756–779.

[9] B. De Meester, P. Heyvaert, A. Dimou and R. Verborgh, Towards a Uniform User Interface for Editing Data Shapes, in: *Proceedings of the 4th International Workshop on Visualization and Interaction for Ontologies and Linked Data*, Vol. 2187, 2018, pp. 13–24. ISSN 1613-0073.

[10] J.E.L. Gayo, D. Fernández-Álvarez and H. García-González, RDFShape: An RDF Playground Based on Shapes, in: *International Semantic Web Conference*, 2018.

[11] E. Prud'hommeaux, J.E. Labra Gayo and H. Solbrig, Shape expressions: an RDF validation and transformation language, in: *Proceedings of the 10th International Conference on Semantic Systems*, New York, NY, United States, 2014, pp. 32–40.

[12] S. Lieber, B. De Meester, A. Dimou and R. Verborgh, Statistics about Data Shape Use in RDF Data, in: *Proceedings of the 19th International Semantic Web Conference: Posters, Demos, and Industry Tracks*, CEUR Workshop Proceedings, Vol. 2721, 2020, pp. 330–335. ISSN 1613-0073.

[13] OMG, Unified Modeling Language, Version 2.5.1, Technical Report, Object Management Group, 2017. https://www.omg.org/spec/UML/2.5.1/.

[14] S. Cranefield and M. Purvis, UML as an Ontology Modelling Language, in: *Intelligent Information Integration*, 1999.

[15] OMG, Ontology Definition Metamodel, Version 1.1, Technical

Report, Object Management Group, 2014. https://www.omg.org/spec/ODM/1.1.

[16] H. Knublauch, J.A. Hendler and K. Idehen, SPIN – Overview and Motivation, Member Submission, World Wide Web Consortium (W3C), 2011. https://www.w3.org/Submission/spin-overview/.

[17] A. Ryman, Resource Shape 2.0, Member Submission, World Wide Web Consortium (W3C), 2014. https://www.w3.org/Submission/shapes/.

[18] A.G. Ryman, A.J. Le Hors and S. Speicher, OSLC Resource Shape: A language for defining constraints on Linked Data, in: *Proceedings of the WWW2013 Workshop on Linked Data on the Web*, Vol. 996, 2013.

[19] E. Prud'hommeaux, Shape Expressions 1.0 Primer, Member Submission, World Wide Web Consortium (W3C), 2014. https://www.w3.org/Submission/2014/SUBM-shex-primer-20140602/.

[20] H. Knublauch, From SPIN to SHACL, Technical Report, 2017. https://spinrdf.org/spin-shacl.html.

[21] T. Bosch, A. Nolle, E. Acar and K. Eckert, RDF Validation Requirements – Evaluation and Logical Underpinning, 2015.

[22] S. Lieber, B. De Meester, A. Dimou and R. Verborgh, MontoloStats – Ontology Modeling Statistics, in: *Proceedings of the 10th International Conference on Knowledge Capture - K-CAP '19*, 2019, pp. 69–76.

[23] S. Steyskal and K. Coyle, SHACL Use Cases and Requirements, Technical Report.

[24] D. De Paepe, G. Thijs, R. Buyle, R. Verborgh and E. Mannens, Automated UML-Based Ontology Generation in OSLO^2, in: *The Semantic Web: ESWC 2017 Satellite Events – ESWC 2017*, Vol. 10577, 2017, pp. 93–97.

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

[26] F.J. Ekaputra and X. Lin, SHACL4P: SHACL constraints validation within Protégé ontology editor, in: *2016 International Conference on Data and Software Engineering (ICoDSE)*, 2016.

[27] I. Boneva, J. Dusart, D. Fernández Alvarez and J.E.L. Gayo, Shape Designer for ShEx and SHACL Constraints, in: *ISWC 2019 Satellites*, 2019.

[28] C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler and M. Smith, OWL 2 Web Ontology Language – Structural Specification and Functional-Style Syntax (Second Edition), Recommendation, World Wide Web Consortium (W3C), 2012. http://www.w3.org/TR/owl2-syntax/.

[29] OMG, Object Constraint Language, Version 2.4, Technical Report, Object Management Group, 2014. https://www.omg.org/spec/OCL/2.4.

[30] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis and E. Giannopoulou, Ontology visualization methods – a survey, *ACM Comput. Surv.* **39** (2007), 10.

[31] S. Mikhailov, M. Petrov and B. Lantow, Ontology Visualization: A Systematic Literature Analysis, in: *BIR Workshops*, 2016.

[32] N. Achich, B. Bouaziz, A. Algergawy and F. Gargouri, Ontology Visualization: An Overview, in: *ISDA*, 2017.

[33] A. Anikin, D. Litovkin, M. Kultsova, E. Sarkisova and T. Petrova, Ontology Visualization: Approaches and Software Tools for Visual Representation of Large Ontologies in Learning, in: *Creativity in Intelligent Technologies and Data Science*, 2017, pp. 133–149. ISBN 978-3-319-65551-2.

[34] M. Dudáš, S. Lohmann, V. Svátek and D. Pavlov, Ontology visualization methods and tools: a survey of the state of the art, *The Knowledge Engineering Review* **33** (2018).

[35] M.F. Joseph and R. Lourdusamy, Feature analysis of ontology visualization methods and tools, *Computer Science and Information Technologies* **1**(2) (2020), 61–77.

[36] F. Ghorbel, N. Ellouze, F. Métais, F. Gargouri, N. Herradi et al., MEMO GRAPH: an ontology visualization tool for everyone, *Procedia Computer Science* **96** (2016), 265–274.

[37] G. Braun, C. Gimenez, L. Cecchi and P. Fillottrani, crowd: A Visual Tool for Involving Stakeholders into Ontology Engineering Tasks, *KI - Künstliche Intelligenz* (2020), 1–7.

[38] D. Garijo, WIDOCO: a wizard for documenting ontologies, in: *International Semantic Web Conference*, 2017, pp. 94–102.

[39] S. Lohmann, V. Link, E. Marbach and S. Negru, WebVOWL: Web-based visualization of ontologies, in: *International Conference on Knowledge Engineering and Knowledge Management*, 2014, pp. 154–158.

[40] S. Lohmann, S. Negru and D. Bold, The ProtégéVOWL plugin: ontology visualization for everyone, in: *European Semantic Web Conference*, 2014, pp. 395–400.

[41] J.W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 3rd edn, Sage Publications Ltd., 2008.

[42] R. Likert, A technique for the measurement of attitudes., *Archives of psychology* **22**(140) (1932), 55.

[43] S.S. Shapiro and M.B. Wilk, An analysis of variance test for normality (complete samples), *Biometrika* **52**(3/4) (1965), 591–611.

[44] F. Wilcoxon, Individual comparisons by ranking methods, in: *Breakthroughs in statistics*, Springer, 1992, pp. 196–202.

[45] J. Recker, *Scientific Research in Information Systems: A Beginner's Guide*, Springer Science & Business Media, 2012.

[46] World Wide Web Consortium (W3C), Semantic Web - Vocabularies, 2009.

[47] I. Boneva, J. Dusart, D. Fernández-Álvarez and J.E. Emilio Labra Gayo, Semi Automatic Construction of ShEx and SHACL Schemas, *CoRR* **abs/1907.10603** (2019).

[48] R. Cyganiak, D. Wood and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, Recommendation, World Wide Web Consortium (W3C), 2014. http://www.w3.org/TR/rdf11-concepts/.

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

[50] B. De Meester, P. Heyvaert and A. Dimou, YARRRML, Unofficial Draft, imec – Ghent University – IDLab, 2019. https://w3id.org/yarrrml/spec.

[51] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van de Walle, 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, 2014. ISSN 16130073.

[52] R. Falco, A. Gangemi, S. Peroni, D. Shotton and F. Vitali, Modelling OWL Ontologies with Graffoo, in: *The Semantic Web: ESWC 2014 Satellite Events*, Vol. 8798, 2014, pp. 320–325.

[53] D. Fernández-Álvarez, H. García-González, J. Frey, S. Hell-

mann and J.E.L. Gayo, Inference of Latent Shape Expressions Associated to DBpedia Ontology, in: *International Semantic Web Conference*, 2018.

[54] H.-G. Fill, B. Pittl and G. Honegger, A modeling environment for visual SWRL rules based on the SeMFIS platform, in: *International Conference on Design Science Research in Information System and Technology*, 2017, pp. 452–456.

[55] P. Heyvaert, A. Dimou, R. Verborgh, E. Mannens and R. Van de Walle, Towards Approaches for Generating RDF Mapping Definitions, in: *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, Vol. 1486, 2015. ISSN 1613-0073.

[56] G.A. Miller, The magical number seven, plus or minus two: Some limits on our capacity for processing information., *Psychological review* **63**(2) (1956), 81.

[57] C. Pinkel, C. Binnig, P. Haase, C. Martin, K. Sengupta and J. Trame, How to Best Find a Partner? An Evaluation of Editing Approaches to Construct R2RML Mappings, in: *The Semantic Web: Trends and Challenges*, 2014, pp. 675–690.

[58] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language Reference Manual (2nd Edition)*, Pearson Higher Education, 2004. ISBN 0321245628.

[59] B. Spahiu, A. Maurino and M. Palmonari, 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)*, 2018, pp. 103–117.

[60] V. Wiens, S. Lohmann and S. Auer, GizMO–A Customizable Representation Model for Graph-Based Visualizations of Ontologies, in: *Proceedings of the 10th International Conference on Knowledge Capture*, 2019, pp. 163–170.