Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

A Library for Visualizing SHACL over Knowledge Graphs

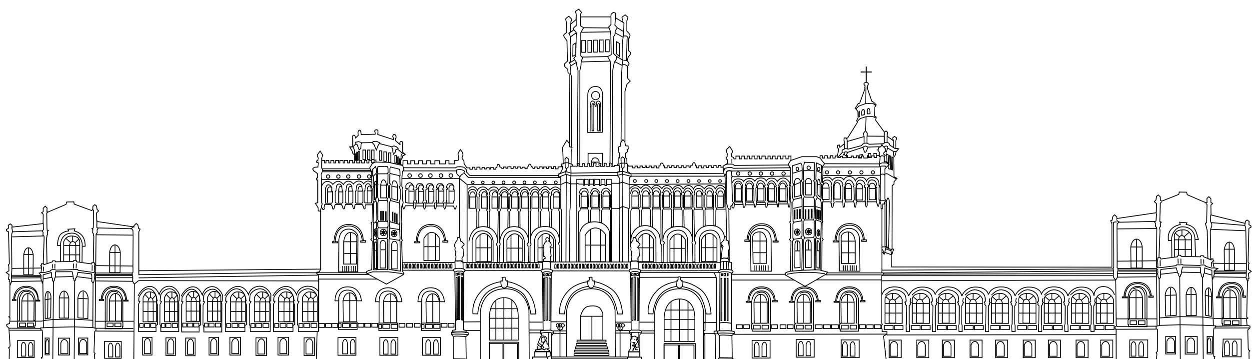
A thesis submitted in fulfillment of the requirements for the degree of
Master of Science in Internet Technologies and Information Systems (ITIS)

BY

Hany Alom
Matriculation number: 10009810
E-mail: hany.alom@gmail.com

First evaluator: Prof. Dr. Maria-Esther Vidal
Second evaluator: Prof. Dr. Sören Auer
Supervisor: M.Sc. Philipp D. Rohde

March 24, 2022



Declaration of Authorship

I, Hany Alom, declare that this thesis titled, 'A Library for Visualizing SHACL over Knowledge Graphs' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Hany Alom

Signature: ______________________________

Date: ______________________________

Acknowledgements

First, I would like to thank Prof. Dr. Sören Auer and Prof. Dr. Maria-Esther Vidal for giving me the opportunity to develop my thesis at TIB. I would like to express my sincere gratitude to Prof. Dr. Maria-Esther Vidal, for her continuous moral and scientific support throughout the project. I also truly appreciate my supervisor, M.Sc. Philipp D. Rohde, for all the effort and time spent supervising the progress and guiding me. Special thanks to my parents and all family members, without whom I could not have the constant motivation. Thanks to my friends for being patient with me and keeping me company during my studies and staying with me up nights.

Hany Alom

Abstract

In a data-driven world, the amount of data currently collected and processed is perhaps the most spectacular result of the digital revolution. And the range of possibilities available has grown and will continue to grow. The Web is full of documents for humans to read, and with Semantic Web, data can also be understood by machines. W3C standardized RDF to represent the Web of data as modeled entities and their relations. Then SHACL came along to present constraints in RDF knowledge graphs, as a network of shapes. SHACL networks are usually presented in textual formats. This thesis focuses on visualizing SHACL networks in a 3D space, while providing many features for the user to manipulate the graph and get the desired information. Thus, *SHACLViewer* is presented as a framework for SHACL visualization. In addition, an evaluation for the impact of various parameters like network size, topology, and density are studied. During the study, execution times for different functions are computed; they include loading time, expanding the graph, and highlighting a shape. The observed results reveal the characteristics of the SHACL networks that affect the performance and scalability of *SHACLViewer*.

Keywords: Knowledge Graph, SHACL, Ontology Visualization

Contents

1 Introduction **1**
1.1 Motivating Example 1
1.2 Contributions 4
1.3 Overview of the Document 4
1.4 Summary of the Chapter 4

2 Background **5**
2.1 Semantic Web Technologies 5
2.2 Resource Description Framework 5
2.3 Web Ontology Language 6
2.4 Shapes Constraint Language 7
2.5 Knowledge Graph 7
2.6 JavaScript Object Notation 7
2.7 Summary of the Chapter 8

3 Related Work **9**
3.1 GizMO 9
3.2 Modelling OWL Ontologies with Graffoo 10
3.3 Ontology visualization methods and tools: a survey of the state of the art 11
3.4 Visualizing ontologies with VOWL 12
3.5 SHAPEness: a SHACL-driven RDF Graph Editor 13
3.6 Summary of the Chapter 14

4 SHACLViewer **15**
4.1 Proposed Architecture 15
4.2 Features 15
4.2.1 Expand and Collapse 16
4.2.2 Search 16

4.2.3 Expand and Collapse All 17
4.2.4 Hide Intra-Constraints 17
4.2.5 Highlight Selected 17
4.2.6 Show Selected Shape Only 17
4.2.7 Center Graph and Focus Selected Shape 17
4.2.8 Link Length 17
4.2.9 Light and Dark Mode 18
4.2.10 Info Panel 18
4.2.11 Shapes Checklist Panel 18
4.3 Summary of the Chapter 19

5 Implementation 20
5.1 Processing SHACL Shape Schemas 20
5.1.1 Server Side 20
5.1.2 Client Side 21
5.2 Visualizing SHACL Shape Schemas 24
5.3 Exploring SHACL Shape Schemas 24
5.3.1 Expand and Collapse 25
5.3.2 Search 25
5.3.3 Expand and Collapse All 26
5.3.4 Hide Intra-Constraints 26
5.3.5 Highlight Selected 26
5.3.6 Show Selected Shape Only 27
5.3.7 Light and Dark Mode 27
5.3.8 Info Panel 28
5.3.9 Shape Checklist Panel 28
5.3.10 *force-graph* Supported Functions 29
5.4 Summary of the Chapter 30

6 Experimental Evaluation 32
6.1 Experimental Setup 32
6.1.1 Benchmarks 32
6.1.2 Metrics 36
6.1.3 Setup 36
6.2 Impact of Shape Schema Size 37
6.3 Impact of Shape Schema Topology 38
6.4 Impact of Shape Schema Density 39
6.5 Real-World Shape Schemas 39

6.6 Summary of the Chapter 41

7 Conclusions and Future Work **43**
7.1 Conclusions 43
7.2 Limitations 44
7.3 Future Work 44
7.4 Summary of the Chapter 44

Bibliography **45**

List of Figures

1.1 Motivation Example 3

3.1 the visual characteristics of the two different methods and tools, visual appearance and spatial arrangement for *GizMO* [26] 9
3.2 *Graffoo* palette in yEd. [26] 10
3.3 Table of interaction techniques provided by the reviewed tools. [11] 11
3.4 A small ontology visualized with *VOWL* [21] 12
3.5 *SHAPEness* user interface, consisting of: Graph View (a), Palette View (b), Outline View (c), Properties View (d), RDF/Turtle View (e) [22] 14

4.1 SHACLViewer Architecture 16

5.1 *SHACLViewer*. a screen-shoot of *SHACLViewer* full web-page 21
5.2 *Trav-SHACL*JavaScript file 22
5.3 The visual representation in *SHACLViewer*, there are to types of constraints: inter-constraint with two states and intra-constraint. And four colors for nodes depending on the node's type and state. Also, nodes are grouped by shape and each group has a random color assigned to it. 24
5.4 Expand and Collapse 25
5.5 Search box 26
5.6 Expand All and Collapse All 27
5.7 Hide Intra-Constraints 28
5.8 Show Selected Shape Only 29
5.9 Light and Dark Mode 30
5.10 **Info Panel** This panel shows information about the selected shape and a tree view for its constraints. 31
5.11 **Shape Checklist Panel** This panel shows all shapes listed in a check list where the user could show or hide any shape. 31
5.12 **Options Menu** This menu contains most of the features supported by *SHACLViewer*. 31

6.1 Visualization of synthetic graphs with SHACLViewer 35

6.2 Size Impact on Execution Time 37
6.3 Impact of Number of Nodes on Execution Time 38
6.4 Impact of Number of Links on Execution Time 39
6.5 Topology Impact on Execution Time 40
6.6 Density Impact on Execution Time 40
6.7 Density Impact on Execution Time with number of Nodes and Links 41
6.8 Real-World Shape Schemas Size 41
6.9 Execution Time on Real-World Shape Schemas 42

List of Tables

6.1 Synthetic SHACL Shape Schemas 34
6.2 Statistics of Real-World SHACL Shape Schemas 36

Acronyms

JSON JavaScript Object Notation

KG Knowledge Graph

OWL Web Ontology Language

RDF Resource Description Framework

SHACL Shapes Constraint Language

Chapter 1

Introduction

Knowledge graphs have gained momentum as data structures that enable the representation of data and metadata as factual statements[16]. The Semantic Web community has made several formalisms and tools available to facilitate the implementation and publication of knowledge graphs. Specifically, the World Wide Web Consortium (W3C) has proposed standards and recommendation languages to allow for the interoperability of knowledge graphs. These languages include: the Resource Description Framework (RDF) [19], the Web Ontology Language (OWL) [23], and the Shapes Constraint Language (SHACL) [17]. RDF is a data model for representing knowledge graphs based on their entities and their relations. OWL is built on top of RDF and Description Logics to represent complex relations between entities. Lastly, SHACL allows expressing integrity constraints over RDF knowledge graphs. In SHACL, restrictions are described as a network of shapes (a.k.a. SHACL shape schema). SHACL networks can be expressed in RDF or textual format, impacting the readability of the represented constraints. This thesis presents *SHACLViewer*, a tool to visualize SHACL networks with many valuable features. This work is motivated by the following example.

1.1 Motivating Example

To motivate this thesis work, consider a SHACL network consisting of shapes, those shapes are in the form of JSON files. Each JSON file contains the name of the shape and an array of constraints of two types, the first one is called intra-constraint which declares a local restriction that does not target another shape like name or phone number. The second is called inter-constraint which is a constraint that targets another shape like a teacher teaches-in school. That means the shapes are connected

by inter-constraints and forming a network that can be visualized. For this example let us check these five shapes written as JSON files:

```
{"name": "University",
 "targetDef": {
  "query":"SELECT ?x WHERE {?x a ub:University}",
  "class": "ub:University"
 },
 "prefix": {
  "ub": "<http://swat.cse.lehigh.edu/onto/univ-bench.owl#>"
 },
 "constraintDef": {
  "conjunctions": [
   [
    { "path": "ub:name", "min": 1, "max": 1}
   ]
  ]
 }
}
```

```
{"name": "Department",
 "targetDef": {
  "query":"SELECT ?x WHERE {?x a ub:Department}",
  "class": "ub:Department"
 },
 "prefix": {
  "ub": "<http://swat.cse.lehigh.edu/onto/univ-bench.owl#>",
  "blah": "<http://some.prefix.com/blah/>"
 },
 "constraintDef": {
  "conjunctions": [
   [
    { "path": "ub:name", "min": 1, "max": 1},
    { "path": "ub:subOrganizationOf", "min": 1, "max": 1, "shape": "University" }
   ]
  ]
 }
}
```

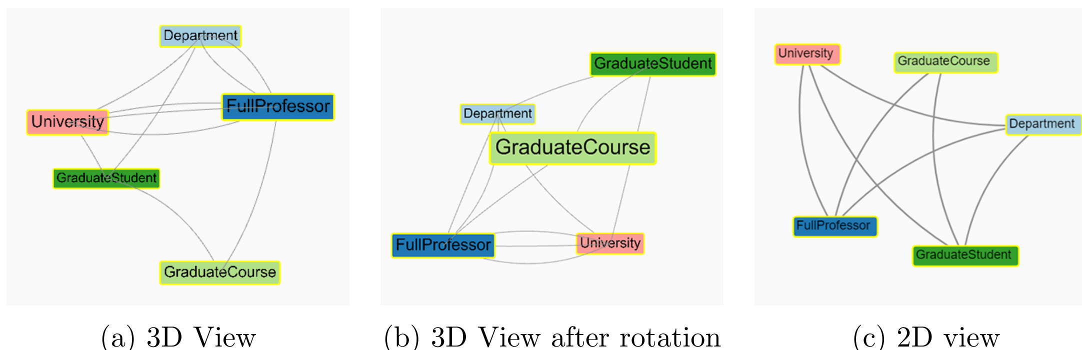
```
{"name": "GraduateCourse",
 "targetDef": {
  "query":"SELECT ?x WHERE {?x a ub:GraduateCourse}",
  "class": "ub:GraduateCourse"
 },
 "prefix": {
  "ub": "<http://swat.cse.lehigh.edu/onto/univ-bench.owl#>"
 },
 "constraintDef": {
  "conjunctions": [
   [
    { "path": "ub:name", "min": 1, "max": 1}
   ]
  ]
 }
}
```

```
{"name": "FullProfessor",
 "targetDef": {
  "query":"SELECT ?x WHERE {?x a ub:FullProfessor}",
  "class": "ub:FullProfessor"
 },
 "prefix": {
  "ub": "<http://swat.cse.lehigh.edu/onto/univ-bench.owl#>"
 },
 "constraintDef": {
  "conjunctions": [
   [
    { "path": "ub:doctoralDegreeFrom", "min": 1, "shape": "University" },
    { "path": "ub:emailAddress", "min": 1 },
    { "path": "ub:headOf", "max": 1, "shape": "Department" },
    { "path": "ub:mastersDegreeFrom", "min": 1, "shape": "University" },
    { "path": "ub:name", "min": 1, "max": 1},
    { "path": "ub:researchInterest", "min": 1},
```

```
    { "path": "ub:teacherOf", "min": 1, "shape": "GraduateCourse" },
    { "path": "ub:teacherOf", "max": 2, "shape": "GraduateCourse" },
    { "path": "ub:telephone", "min": 1 },
    { "path": "ub:undergraduateDegreeFrom", "min": 1, "shape": "University" },
    { "path": "ub:worksFor", "min": 1, "shape": "Department" }
   ]
  ]
 }
}
```

```
{"name": "GraduateStudent",
 "targetDef": {
  "query":"SELECT ?x WHERE {?x a ub:GraduateStudent}",
  "class": "ub:GraduateStudent"
 },
 "prefix": {
  "ub": "<http://swat.cse.lehigh.edu/onto/univ-bench.owl#>"
 },
 "constraintDef": {
  "conjunctions": [
   [
    { "path": "ub:advisor", "min": 1, "max": 1 },
    { "path": "ub:emailAddress", "min": 1 },
    { "path": "ub:memberOf", "min": 1, "shape": "Department" },
    { "path": "ub:name", "min": 1, "max": 1},
    { "path": "ub:takesCourse", "min": 1, "shape": "GraduateCourse" },
    { "path": "ub:takesCourse", "max": 3, "shape": "GraduateCourse" },
    { "path": "ub:telephone", "min": 1 },
    { "path": "ub:undergraduateDegreeFrom", "min": 1, "shape": "University" }
   ]
  ]
 }
}
```

By using *SHACLViewer*, this network could be visualized in 3D Figure 1.1a and 2D views Figure 1.1c, that supports a list of useful features like expanding and collapsing shapes, selecting a shape and checking its information, hiding other shapes and more.



(a) 3D View (b) 3D View after rotation (c) 2D view

Figure 1.1: **Motivation Example**: a) 3D view of the network, b) the same view after dragging the background rotating the graph, c) 2D view of the same network.

1.2 Contributions

This thesis contributes to the repertoire of tools for knowledge graph management and presents *SHACLViewer*, a visualization tool for SHACL schemas. *SHACLViewer* supports 3D/2D visualizations and enables the visualization of essential features without sacrificing the usability of the tool. The performance of *SHACLViewer* is empirically evaluated in a benchmark of synthetic and real-world SHACL networks. The results of the experimental study reveal the conditions of the SHACL schemas that impact the performance and scalability of *SHACLViewer*. These conditions include the topology, size, and density of the graph.

1.3 Overview of the Document

The remainder of this document is structured as follows: Chapter 2 provides the theoretical background needed to understand the approach and methods proposed in this thesis. Related work is presented in Chapter 3. Chapter 4 introduces the architecture and formalization of the proposed approach. Chapter 5 describes in more detail the implementation of the approach. A scalability study of the implemented approach is presented, discussed, and analyzed in Chapter 6. Finally, Chapter 7 concludes this thesis and presents ideas to be addressed as future work on the topic of visualizing SHACL shape schemas.

1.4 Summary of the Chapter

To summarize, this chapter introduces the working area of this thesis. It also provides a motivating example to understand the existing problem better that this work aims to tackle. In addition, this chapter describes the contribution of this thesis.

Chapter 2

Background

This chapter presents the terminology and concepts required to understand the problem tackled in this thesis.

2.1 Semantic Web Technologies

The Semantic Web enhances the Web of documents with the possibility of representing data and their meaning (a.k.a. metadata). Tim Berners-Lee is the inventor of the Web, and in a seminal paper published in 2001, Berners-Lee et al. devise the extension of the Web with meaning [7]. The Semantic Web comprises a set of formalisms and technologies that make possible the definition of the semantic layer. The Resource Description Framework (RDF) [19], the Web Ontology Language (OWL) [23], and Shapes Constraint Language (SHACL) [17] correspond to formalisms for modeling data and their integrity constraints.

2.2 Resource Description Framework

The Resource Description Framework (RDF) [19] is the W3C (World Wide Web Consortium) standard for the representation of Web data as factual statements. RDF allows for modeling entities and their relationships using subject-property-object statements, named RDF triples [19]. An RDF t triple is defined as *t=(subject, property,object)*:

- *subject*: corresponds to a resource modeling entities of any type. For example, the professors Soren Auer and Maria-Esther Vidal are resources.

- *property*: represents the predicate that relates a subject with an object. For example, the property *lectures* that relates a professor with the lecture that he/she teaches.

- *object*: models an entity or value that is related to the subject. For example, the course *Knowledge Engineering and Semantic Web* is an object of the property *lectures* when the subjects are Soren Auer or Maria-Esther Vidal.

2.3 Web Ontology Language

The World Wide Web Consortium (W3C) has defined a more descriptive Web Ontology Language (OWL) [23] to extend the restricted expressiveness of RDF. OWL is a new formal language for describing ontologies in the Semantic Web. OWL incorporates elements from multiple representation language families, especially Description Logics and Frames. OWL provides the following features [2]:

- Local scope of properties, e.g., the range of a property.

- Disjointedness of classes: declaring two classes are disjoint.

- Boolean combinations of classes: combining two classes using union.

- Cardinality restrictions: this constraint restricts on how many values a property could have.

- Special characteristics of properties: like a property could be unique, inverse or transitive.

OWL is a very advanced language. As for the example in Section 2.2, the following statements can be defined:

- *Disjointedness:* University and Student are disjoint classes.

- *Different Individuals:* Database and Networking are distinct individuals.

- *Inverse properties: hasLecture* and *TeachedIn* are inverse properties.

- *Membership: SuspendedStudent* is defined as the members of Student that have no property of *Study*.

2.4 Shapes Constraint Language

The SHapes Constraint Language (SHACL) [17] is the W3C recommendation language to represent constraints in RDF. In SHACL, constraints are expressed as a network of shapes (a.k.a. SHACL shape schema). A shape represents integrity constraints over the properties of an RDF term (e.g., class or entities). On the other hand, an edge between two shapes expresses constraints against the properties between the RDF terms associated with the related shapes. A *focus node* represents an RDF triple subject; it can be provided as input or extracted from *target declarations*. A *target* corrresponds to a triple where the subject is the shape name and the property correspond to any of the following predicates: i) a target node `sh:targetNode`; ii) a class `sh:targetClass`; iii) subjects of a predicate `sh:targetSubjectsOf`; and iv) objects of a predicate `sh:targetObjectsOf`. The `sh:path` parameter states a predicate that connects a focus node with its property value. Cardinality constraints are represented with the parameters `sh:minCount` and `sh:maxCount`.

2.5 Knowledge Graph

Knowledge graphs (KGs) are data structures that represent factual knowledge as entities and their relationships using a graph data model [25]. Metadata is part of the KG as well as taxonomies of entities, relationships, and classes. Ontologies and controlled vocabularies are utilized to describe the meaning of the relations, as well as for annotating entities in a uniform way in the knowledge graph. Thus, KGs contribute to the development of a common understanding of the meaning of entities in a domain, and provide a formal specification of the meaning of these entities.

2.6 JavaScript Object Notation

JSON is a lightweight, text-based, language-independent data interchange format; meaning it is a text format used to exchange data between platforms, and being independent of any programming language is helping with that. JSON is a text format for a serialized object of structured data where strings, integers, booleans, and null are the four primitive types that JSON could represent, as well as two structured types (objects and arrays) [6] [10].

2.7 Summary of the Chapter

This chapter presented all the concepts required to understand the problem addressed in this thesis, the proposed approach, and the empirical evaluation conducted to analyze the scalability of the implementation of the approach.

Chapter 3

Related Work

Visualization-related topics have previously been covered in the literature. This chapter discusses state-of-the-art approaches for visualizing ontologies, SHACL schemas, and knowledge graphs. The main features of each system are discussed, and *SHACLViewer* is positioned as a tool that can overcome these approaches' limitations.

3.1 GizMO

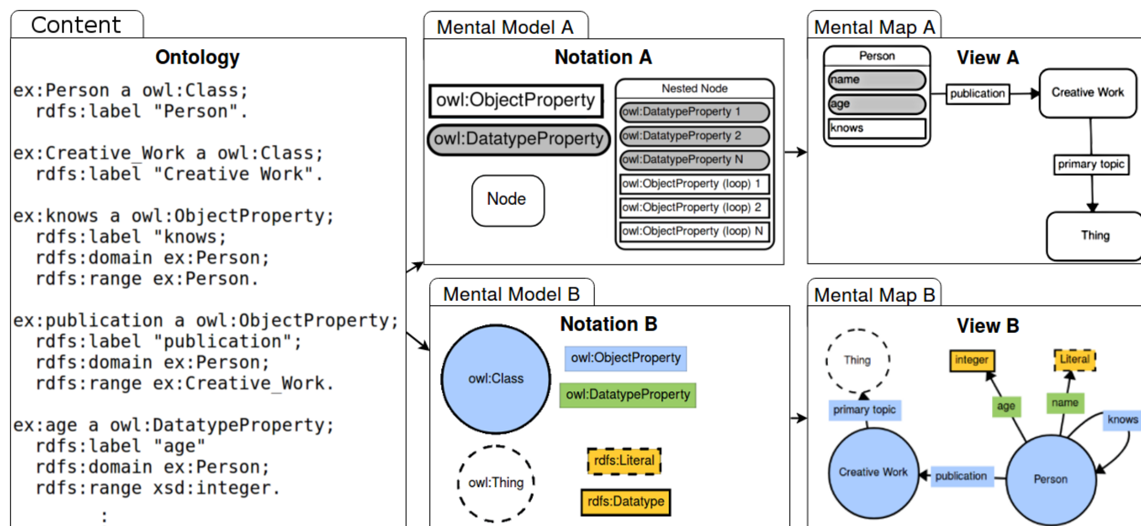


Figure 3.1: the visual characteristics of the two different methods and tools, visual appearance and spatial arrangement for *GizMO* [26]

GizMO, a representation model for graph-based ontology visualizations in 2D space Figure 3.1, demonstrates the methodology's applicability. Annotation properties for visual attributes are defined in the GizMO core ontology (e.g., shapes, colors, positions, etc.). Annotation objects provide grouped instantiations of values connected to OWL constructs and domain ontology elements.

Annotation objects come in a variety of shapes and sizes, each focusing on a different component of the visualization. For the visual representation, these offer a conceptual boundary between the global and local layers [26]. GizMO allows multiple links between two nodes to collapse and expand. These mechanisms for datatypes, datatype properties, and object properties are used to describe nested node visualizations. Where the semantic zooming approach removes collapsed elements from the visualization, the frameworks render the collapsed elements with their visual descriptions inside the corresponding node [26].

3.2 Modelling OWL Ontologies with Graffoo

Graffo is a graphical notation to model OWL ontologies. All the graphical elements blocks "nodes" and arcs "links" Figure 3.2 of Graffoo have been developed using the standard library of yEd [http://www.yworks.com/en/products yed about.html] a free diagram editor [12].

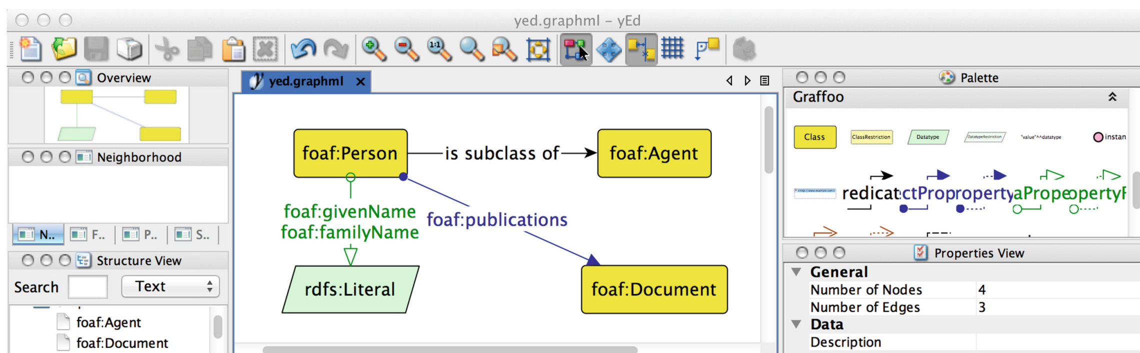


Figure 3.2: *Graffoo* palette in yEd. [26]

It is possible to add annotations to ontological entities by using the preference panel, Graffo supports zooming in and out, panning, entry focus, search and highlight, and drag and drop [11].

3.3 Ontology visualization methods and tools: a survey of the state of the art

This survey evaluates how visualization methods are used and how they are implemented in each tool, as well as the support for interaction techniques and which OWL constructs are shown (what is referred to as OWL coverage) [11]. Figure 3.3 provides an overview of the supported interaction techniques. Definitions for the

	Radar view	Graphical zoom	Entity focus	History (undo/redo)	Pop-up window	Incremental exploration	Search and highlight	Filter parts	Filter entity types	Fisheye distortion	Edge bundling	3D navigation	Panning	Drag and drop	Clustering	Textual editing	Visual editing
CmapTools Ontology Editor		x											x	x		x	x
CropCircles		x															
FlexViz		x					x		x				x	x			
GLOW																	
Graffoo	x	x	x				x						x	x			
GrOWL		x	x				x		x					x		x	x
Jambalaya		x	x		x	x	x	x	x	x				x		x	
KC-Viz		x		x	x	x		x	x				x	x	x	x	
Knoocks		x			x		x				x						
Multi-view ontology visualization							x			x							
NavigOWL	x	x			x		x						x	x			
OLSVis		x	x			x	x						x				
Ontodia		x	x	x	x	x	x	x	x				x	x		x	
OntoGraf		x	x		x	x	x		x				x	x		x	
Ontology visualizer		x	x	x		x	x							x			
OntoRama							x		x	x							
OntoSELF		x							x			x					
OntoSphere			x			x						x	x				
OntoStudio		x	x	x	x	x	x	x	x								
OntoTrack	x	x	x			x	x						x	x		x	x
OntoTrix	x	x	x				x		x		x		x	x	x		
Ontoviewer	x	x					x		x	x			x				
OntoViz		x						x					x				
OWL-VisMod																	
OWLeasyViz		x				x	x		x								
OWLGrEd	x	x		x	x						x			x		x	x
OWLViz		x				x		x								x	
Protégé Entity Browser					x	x	x									x	
SOVA		x					x		x				x	x			
TGViz		x	x			x	x	x	x	x			x				
TopBraid	x	x						x	x				x	x		x	
Triple20			x	x	x		x	x	x							x	x
WebVOWL		x			x		x		x				x	x			

Figure 3.3: Table of interaction techniques provided by the reviewed tools. [11]

mentioned interaction techniques in Figure 3.3:

- Radar view: a visually summarized ontology.
- Graphical zoom: the ability to zoom in and out.
- Entry focus: showing an entity and its surroundings and hiding everything else.
- History: the ability to undo or redo the last step
- Pop-up window: show the details of a selected entity
- Incremental exploration: the ability to add parts of the ontology to the graph after hiding them.
- Search and highlight: the ability to search for an entity.

- Filter parts: hiding parts of the graph based on its type to reduce clutter.
- Filter entity type: hiding all entities of the same type
- Fisheye distortion: focusing on a part of the graph while spreading back the rest.
- Edge bundling: reduce clutter by grouping edges with the same path.
- Panning: moving the graph by dragging.
- Drag and drop: moving an individual entity in the graph.
- Textual editing: the ability to edit the selected entity
- Visual editing: the ability to create new entities.

3.4 Visualizing ontologies with VOWL

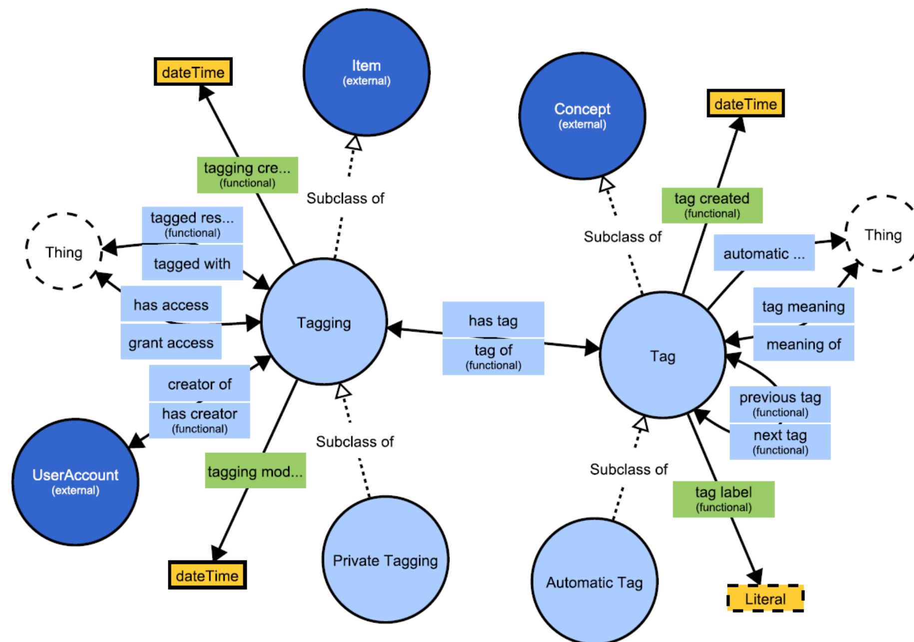


Figure 3.4: A small ontology visualized with *VOWL* [21]

VOWL is a well-defined visual language for representing ontologies in a user-oriented format. It provides graphical representations Figure 3.4 for most Web Ontology Language (OWL) elements that are combined into a force-directed graph layout to visualize the ontology [21]. It supports the following features:

- Layout and navigation: allows the user to zoom in and out and panning.
- The pick-and-pin feature: it allows the user to decouple selected nodes from the force-directed layout and place them on an empty canvas at freely chosen position.
- Node multiplication
- Equivalent classes: merging equivalent classes into one node.
- Properties
- Set operators: special representation for operators like union and intersection.
- Filtering: hide the respective property edges, collapsing and expanding subtrees or sub-graphs
- Colors

3.5 SHAPEness: a SHACL-driven RDF Graph Editor

The fundamental purpose of *SHAPEness* development was to produce a comprehensive desktop program that could be used in any context, domain, or use case that required browsing, editing, or validating RDF graphs based on a set of input SHACL constraints (schema) [22]. There are three types of visualization in *SHAPEness* Figure 3.5: graph-based, form-based and tree-based. The application assists users in properties compilation by hiding the schema's complexity, preventing typos, and validating property types and required properties.

SHAPEness supports 1) graph navigation like zoom in, zoom out, change graph layout, add and remove nodes or relationships 2) hiding a specific node types 3) context menu for frequently used commands.

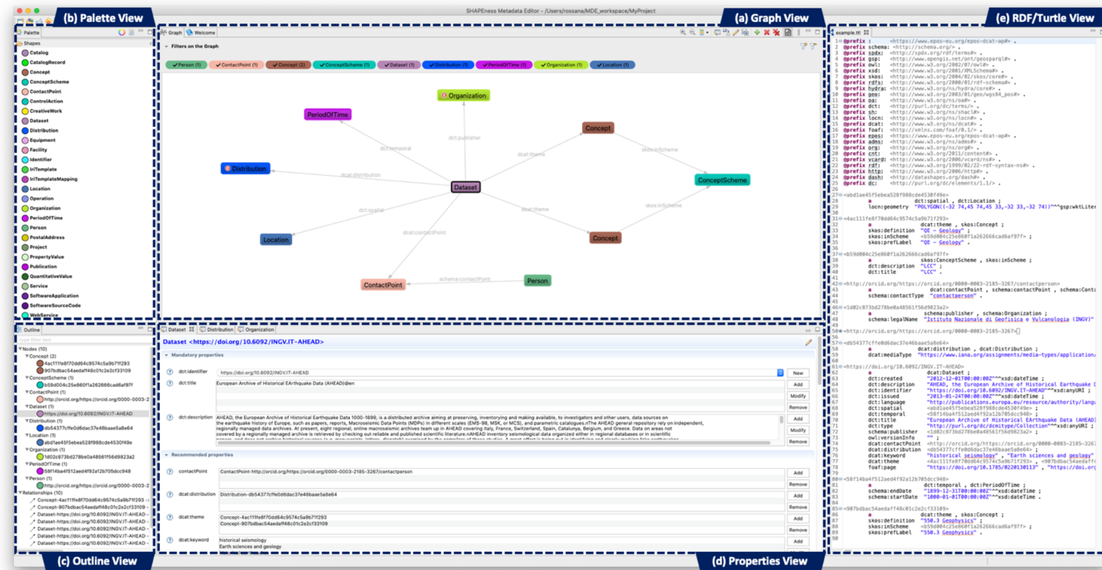


Figure 3.5: *SHAPEness* user interface, consisting of: Graph View (a), Palette View (b), Outline View (c), Properties View (d), RDF/Turtle View (e) [22]

3.6 Summary of the Chapter

This chapter reviewed some work related to this thesis. Additionally, they were compared to the work of this project.

Chapter 4

SHACLViewer

This chapter defines *SHACLViewer* in terms of an abstract architecture, and the operations that can be performed over a SHACL schema.

4.1 Proposed Architecture

The proposed architecture is depicted in Figure 4.1. First the shapes are already in the format of JSON files set in one directory, each file contains one shape which defines the shape name and constraints. The files will be imported and converted to a graph; the graph is a list of shapes, each one has a list of its constraints. Now, the shapes should be converted into nodes and links, each shape has one node carrying its name and other nodes one for each constraint it has. Then, links are connected between each shapes' node and its constraint, and for inter-constraints links will connect them with their targets. Another type of links is added, which connects shapes' nodes with each other if they have an inter-constraint between them. All that was on the server. Next, on the user's side, the nodes and links will be initialized and mapped with each other, then visualized. We propose to draw the graph in a 3D instead of 2D like most other solutions. By adding a new dimension, there is more space to visualize the data. Also, multiple useful functions are included.

4.2 Features

This section presents the features implemented in *SHACLViewer*.

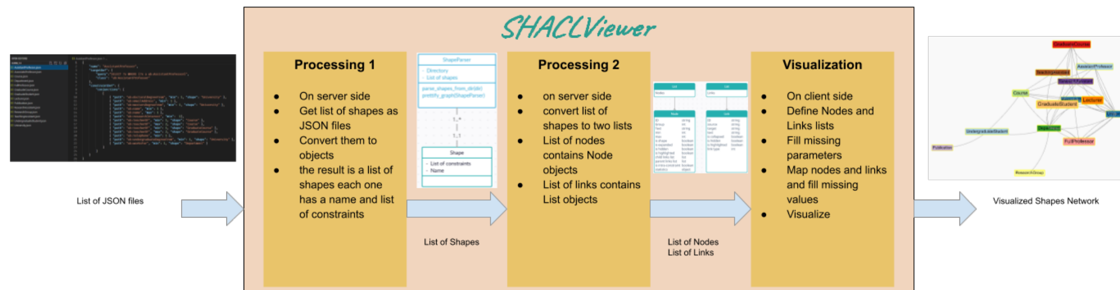


Figure 4.1: **The SHACLViewer Architecture** The diagram reports the architecture of the approach to visualize a shapes network.

4.2.1 Expand and Collapse

Each shape is represented in two states, expanded and collapsed. In the expanded state, a shape is represented with a node carrying the shape name; this node is linked to other nodes representing the constraints. If the constraint is an inter-constraint, the node will be connected to the targeted shape. Contrary, if it is intra-constraint no link is shown. The second state is collapsed, in this state, the shape will be represented with a node carrying the shape name. For each inter-constraint's target, it has there will be a link directly from the shape node to the targeted shape To change between those two states, the user could click on the shape node, and some other features will change the state will be mentioned later.

4.2.2 Search

Using this search box, a user could find any shape. While typing, the user will see suggestions for what he typed and when hovering over one of them the shape will be highlighted in the graph. By clicking on a suggestion, the shape will be selected, and the info panel will be updated. There are some special cases where the selected shape is hidden.

- If the searched shape is hidden by unchecking it from Shape checklist, in this case the shape will be added to the graph and selected.
- When the shape is hidden because of using "Show selected shape only", in this case the searched shape will be selected, and every other shape will be hidden unless it has a connection to the searched shape like if the "Show selected shape only" toggle was turned off and on again.

- Clicking on the search bar will clear it, and suggest all the shapes sorted alphabetically for an easier way to find a shape without typing.

4.2.3 Expand and Collapse All

Two buttons to expand and collapse all shapes, by default all shapes are collapsed, this way when the user opens the graph the load time won't take too much time and the user won't be overwhelmed.

4.2.4 Hide Intra-Constraints

This button will hide all intra-constraints from all expanded shapes. This function is also meant for reducing the clutter in the graph. Default state is off.

4.2.5 Highlight Selected

This toggle hides the highlight colors for the selected shape. Highlighting the selected node may be an intense process if the graph was too big. Default state is on.

4.2.6 Show Selected Shape Only

Turning this switch on will hide all shapes' nodes and links, and will show only the selected shape nodes and links, and it will show any other shape that is related to the selected shape but without its links and constraints. Clicking on a shape's node with a hidden constraint will show them. And any shown node will act as normal.

4.2.7 Center Graph and Focus Selected Shape

Both buttons will change the camera angle of the 3D space to show the graph in different positions. Center graph will put the average nodes' position on the zero axis of the 3D space, while the focus selected shape will set the zero axes on the selected shape main node and rotate the graph to give the node a front view.

4.2.8 Link Length

Those four sliders will increase or decrease the length of the links between the nodes, and the links are separated in four types:

- Shape to Shape link: those links connect shape's node to another shape's node; those links are used to link two shapes if the source shape is collapsed and its constraint linking to the target shape is hidden.
- Shape to Intra-Constraint link: those are links between a shape's main node and an intra-constraint's node, of course the Hide intra-constraints switch should be off to see the change.
- Shape to Inter-Constraint link: those links connect the source shape's node with the inter-constraint's node.
- Inter-Constraint to Shape link: links between inter-constraint's node of a shape and the target shape's node.

4.2.9 Light and Dark Mode

A switch to change the website theme between Light and Dark colors.

4.2.10 Info Panel

This panel is divided into two-part, metadata that contains a number of intra-constraints, inter-constraints and the number of constraints targeting this shape. The second part is the constraints themselves, here the constraints are listed in a tree list starting with the selected shape, in it; first, there are intra-constraints denoted by a symbol, clicking on one of them will show the min/max for that constraint. Second, there are the inter-constraints denoted by another symbol, clicking on one of them will show min/max for that constraint plus the targeted shape by this constraint which can be opened and will show the same thing recursively. By clicking on a shape in a constraint that shape will expand in the graph and show its constraints' nodes, also if the shapes connecting to the clicked shape are hidden, they will be shown again.

4.2.11 Shapes Checklist Panel

In this checklist all shapes in the graph are listed, using this checklist the user could show or hide any shape, also this checklist is updated when the user use "Show selected shape only" or show a shape by clicking on a visible shape with hidden constraints to hidden shapes, or by clicking on a hidden shape name from the constraint tree view in info panel.

4.3 Summary of the Chapter

This chapter presented the visualization problem addressed in this thesis; it defines *SHACLViewer* as an abstract architecture that enables the visualization of SHACL schemas. The implementation of the proposed architecture is presented in Chapter 5, while the results of the empirical study of the performance of the implemented framework are reported in Chapter 6.

Chapter 5

Implementation

For the implementation Python with Flask and Jinja are used. The implementation mostly done in JavaScript but a web application had to be made with Python to use it with other projects. The implemented application is called *SHACLViewer* Figure 5.1.

5.1 Processing SHACL Shape Schemas

5.1.1 Server Side

The SHACL shape schemas used in this project are using the JSON serialization proposed by Corman et al. [9]. Following their proposed serialization, a SHACL shape schema is a folder on the hard drive containing JSON files. Each JSON file represents one shape of the shape schema. The shape files contain the shape name, target definition, and a list of the shape's constraints; each constraint has:

- path: the constraint name
- min/max: for the constraint
- shape: if it is included the constraint is an inter-constraint and it targets a shape, and if it is not, the constraint is intra-constraint

Figuera et al. [14] suggested improvements for the SHACL validation algorithm proposed by Corman et al. [9]. The SHACL validator using the improved algorithm is called *Trav-SHACL*. In this project, the SHACL-JSON parser of Trav-SHACL[1] is used to convert the JSON files!Figure 5.2 into the internal representation of the

[1]https://github.com/SDM-TIB/Trav-SHACL

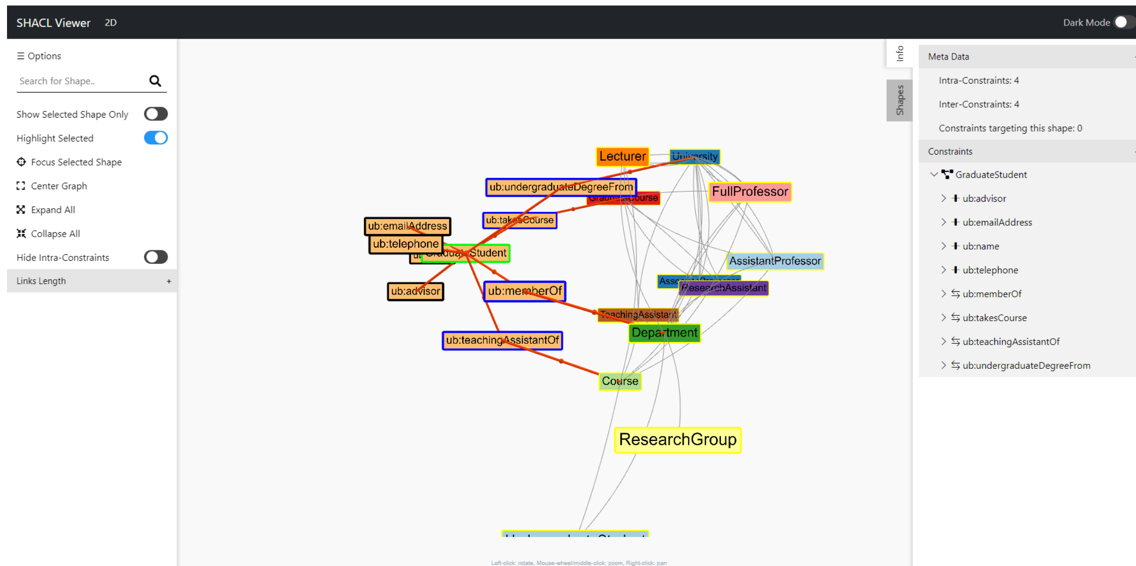


Figure 5.1: *SHACLViewer*. a screen-shoot of *SHACLViewer* full web-page

SHACL shape schema. The list of shapes is injected into the JavaScript code using Jinja; meaning that the data is processed on the server side and the client only needs to read it.

5.1.2 Client Side

The shapes and constraints are already converted to nodes and links while using Jinja from the server side, and on the client side they should be initialized to be represented in a graph.

First, Nodes: there are 3 types of nodes Figure 5.3:

1. shape node: this node represents the shape and carries its name.
2. inter-constraint node: this node has one link connecting it from the shape node to it.
3. intra-constraint node: this node is connected with two links one from the shape node to it and another from it to the targeted shape.

Each node has:

- ID: each node has an ID to be used as a source or target when connecting it with links.

```
{
    "name": "AssistantProfessor",
    "targetDef": {
        "query":"SELECT ?x WHERE {?x a ub:AssistantProfessor}",
        "class": "ub:AssistantProfessor"
    },
    "constraintDef": {
        "conjunctions": [
            [
                { "path": "ub:doctoralDegreeFrom", "min": 1, "shape": "University" },
                { "path": "ub:emailAddress", "min": 1 },
                { "path": "ub:mastersDegreeFrom", "min": 1, "shape": "University" },
                { "path": "ub:name", "min": 1 },
                { "path": "ub:name", "max": 1 },
                { "path": "ub:researchInterest", "min":  1},
                { "path": "ub:teacherOf", "min": 1, "shape": "Course" },
                { "path": "ub:teacherOf", "max": 2, "shape": "Course" },
                { "path": "ub:teacherOf", "min": 1, "shape": "GraduateCourse" },
                { "path": "ub:teacherOf", "max": 2, "shape": "GraduateCourse" },
                { "path": "ub:telephone", "min": 1 },
                { "path": "ub:undergraduateDegreeFrom", "min": 1, "shape": "University" },
                { "path": "ub:worksFor", "min": 1, "shape": "Department" }
            ]
        ]
    }
}
```

Figure 5.2: *Trav-SHACL*JavaScript file

- Group: a string used to color code all nodes in the same group with the same color, and each constraint node has the same group as its shape Figure 5.3 Figure 5.4c.
- Text: the visible text written on the node in the graph.
- min/max: the values for min and max and is added if the node represents a constraint.
- is shape: a Boolean flag to differentiate between a shape node and a constraint node.
- is expanded: a Boolean flag to know if the shape this node is in is expanded or collapsed.
- is hidden: a Boolean flag to know if the node must be shown or hidden in the graph.
- is highlighted: a Boolean flag to know if the node color should change or not. Used in highlighting nodes and links feature.
- child links list: a list of all links that has this node as a source.
- parent links list: a list of all links that has this node as a target.
- is intra-constraint: a Boolean flag to differentiate between an intra-constraint node and an inter-constraint node.

- statistics: an object filled in later state when all nodes and links are created containing number of intra-constraints, inter-constraints, and number of constraints targeting this shape; to be displayed in the Meta Data in the Info Panel.

Then we have the links and we have 4 types of links Figure 5.3:

1. Shape to Shape: this link connects two shapes; this link will be visible when the source shape node is collapsed and the intra-constraints are hidden.
2. Shape to Intra-Constraint: this link connects a shape's node as source with an intra-constraint's node as a target.
3. Shape to Inter-Constraint: this link connects a shape's node as source with an inter-constraint's node as a target.
4. Inter-Constraint to Shape: this link connects an inter-constraint as a source to its targeted shape node as a target.

Each link has:

- ID: each link should have an ID but it has not been used.
- source: the ID for the source node of the link.
- target: the ID for the target node of the link.
- text: a text carrying the constraint name but it has not been used.
- is collapsed: a Boolean flag to know if the shape this node is in is collapsed or expanded.
- is hidden: a Boolean flag to know if the link must be shown or hidden in the graph.
- is highlighted: a Boolean flag to know if the link color should change or not. Used in highlighting nodes and links feature.
- Link Type: a number to differentiate between the four types of links.

Note that there are two flags to show and hide nodes and links, `is expanded/ collapsed` and `is hidden`. This way hiding then showing a shape will preserve its state (expanded or collapsed).

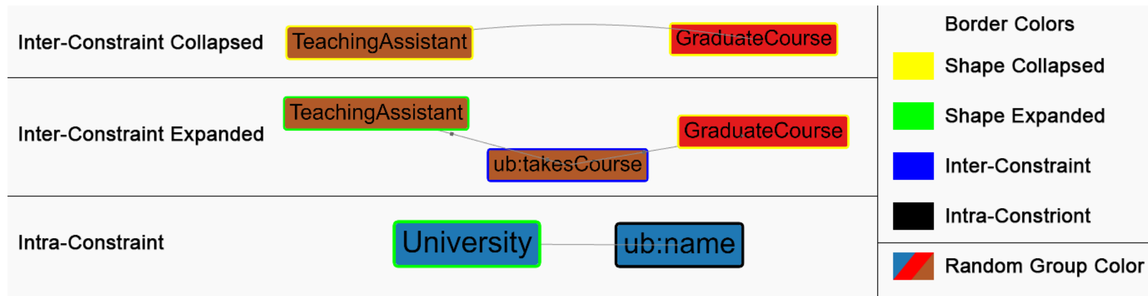


Figure 5.3: The visual representation in *SHACLViewer*, there are to types of constraints: inter-constraint with two states and intra-constraint. And four colors for nodes depending on the node's type and state. Also, nodes are grouped by shape and each group has a random color assigned to it.

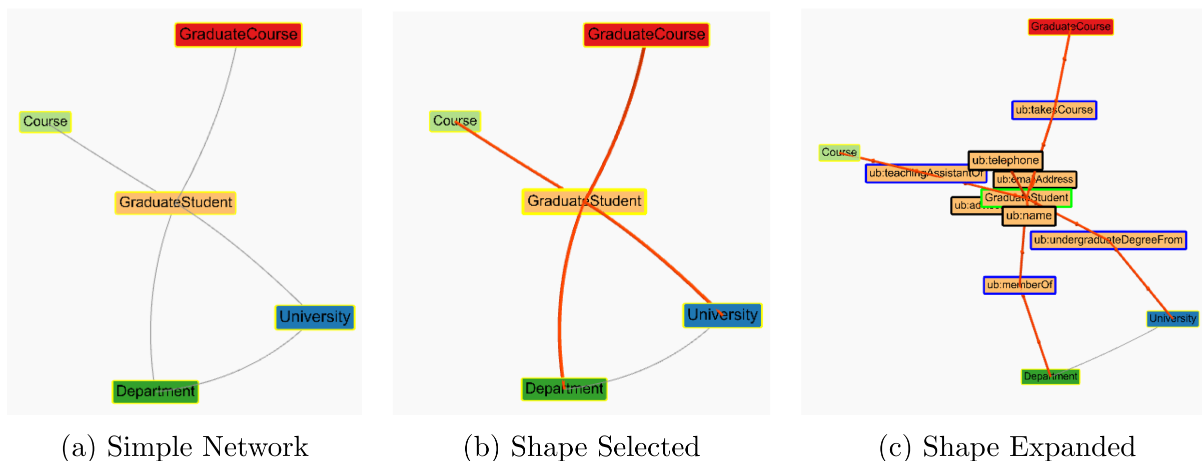
All nodes are saved in a list called Nodes and all links are saved in a list called Links, but searching for a node will take time so a map was introduced called `nodesById` that maps each node to a key which is the node ID, also there is another map for links called `linksById`.

5.2 Visualizing SHACL Shape Schemas

To present the shapes in a 3D graph, the JavaScript library *3d-force-graph* [3] (version 1.70.5) was used. For the 2D visualization, its sister library *force-graph* [4] (version 1.42.4) was used. Both share almost the same functionality; mainly differing in how the nodes of the graph are drawn. Both of them require a list of nodes with IDs, and a list of links with source and target IDs of the nodes to connect between them. Other attributes are for visualization purposes. By providing the main two lists of all nodes and links the graph will show everything including all links that have the same purpose, Shape to Shape links with shape to constraint link → constraint's node → constraint to shape link. So, to hide a link or a node, two new lists where created each time an update on the graph occurs, Visible Nodes and Visible Links. Filling those two new lists with the visible nodes and links then providing them to the graph library will be the solution.

5.3 Exploring SHACL Shape Schemas

Some features have their unique implementation and others share the same concepts.



(a) Simple Network (b) Shape Selected (c) Shape Expanded

Figure 5.4: **Expand and Collapse** Clicking on a selected node will expand it showing its constraints, and clicking it again will collapse it.

5.3.1 Expand and Collapse

First of all, by clicking on any shape's node it will be selected and highlighted without changing its state Figure 5.4b. Then, if the clicked node was previously selected, its state will change from *Collapsed* (the default state) to *Expanded* Figure 5.4c and vice versa. On changing a selected shape status, its connected links should be updated. Depending on the link type its status will change, in case the shape status changed from collapsed to expanded, all links will be set to expanded but not Shape to Shape links, those will be collapsed. Next, the constraints nodes of this shape will be set to expanded, and if they are inter-constraints the constraint to shape link coming out of the constraint's node will be expanded too. Last step is updating the graph, in this function the Visible Nodes and Visible Links lists will be updated after iterating the Nodes and Links lists adding only expanded/not-collapsed nodes and links (also not hidden, used by Show Selected Shape Only feature).

5.3.2 Search

The search box is part of the options menu Figure 5.5. The search function finds only shape nodes, so instead of searching the full Nodes list a new list called suggestions is introduced that contains shapes names only. When the user clicks on the search bar or types something in it, the suggestion list will appear containing a list of shape names according to the user input, or all shape names if the user did not input yet. Hovering on a name will highlight the shape's node, links and constraints nodes

connected to it. Clicking on a name will select the shape in the graph and open the info panel. If the selected shape was hidden by unchecking it from shapes checklist, the shape will be shown and the graph will be updated. Or if show selected shape only is on, its function will be called on the new selected shape.

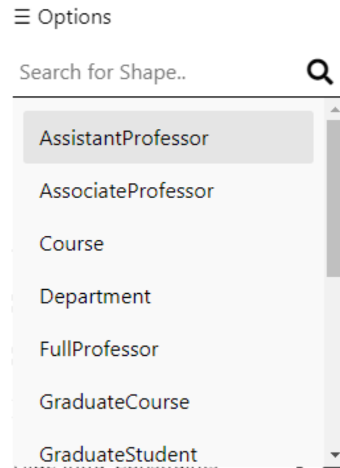


Figure 5.5: **Search box** This search box is located in the option menu, the suggestion panel will show only if the search box is in focus.

5.3.3 Expand and Collapse All

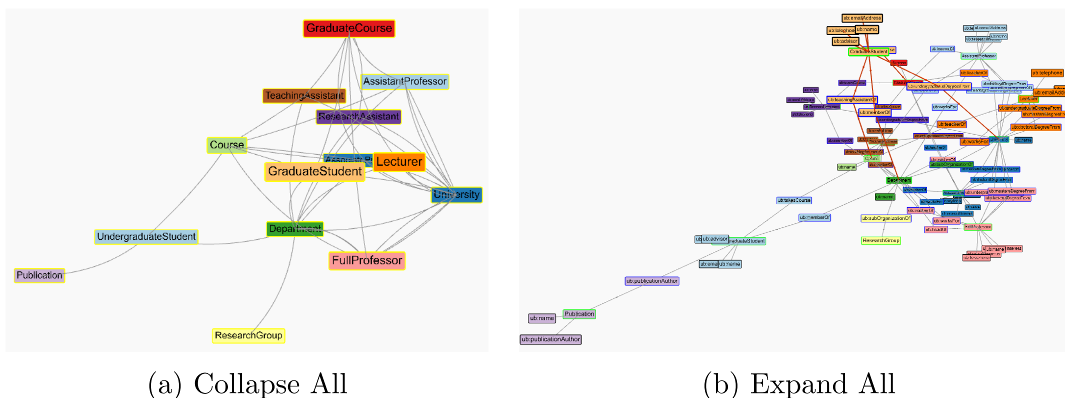
Both functions do the same thing, they iterate Nodes list changing the expanded value for nodes that represents shapes, and iterate the Links list changing the collapsed value depending if the link from type Shape to Shape or not Figure 5.6.

5.3.4 Hide Intra-Constraints

For this feature the HTML toggle will change a Boolean flag which is checked each time the Visible Nodes and Visible Links lists are updated. If the flag is enabled all nodes that are intra-constraints will not be added to the Visible Nodes list, neither intra-constraints links to the Visible Links list Figure 5.7.

5.3.5 Highlight Selected

To highlight a node or a link `is highlighted` value is set to true and an update event is triggered for the graph library to update the colors for nodes and links. Now, when



(a) Collapse All (b) Expand All

Figure 5.6: **Expand All and Collapse All** Using Collapse All will replace all constraints with links, while Expand All will show all constraints nodes for all shapes in the network.

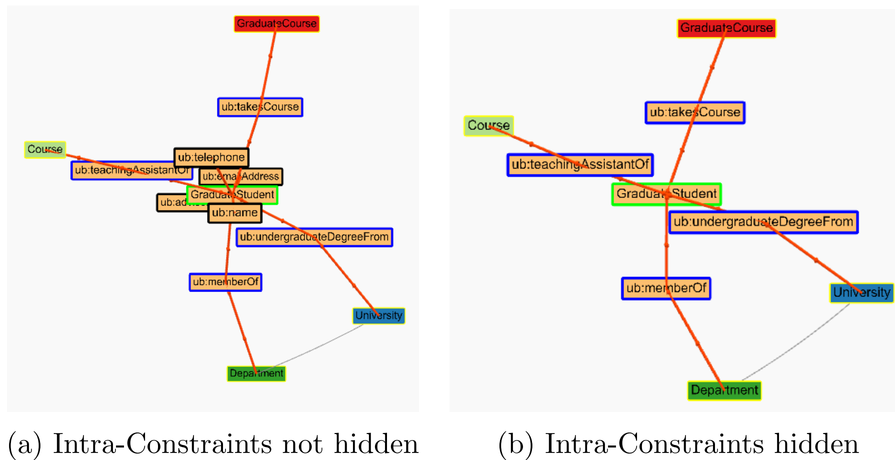
changing a highlighted shape, first all nodes and links `is highlighted` value is set to false. Then, the selected shape node, links and constraints `is highlighted` value is set to true. After words, the update event is triggered for the graph Figure 5.4a Figure 5.4b.

5.3.6 Show Selected Shape Only

When the toggle is turned on, all nodes and links will be hidden and only the selected shape will be shown Figure 5.8 and the Shapes chick list panel will open. A Boolean flag will be set to true that will be used by other functions, for example if the flag is set to true: Search: searching for a shape and selecting it will call the Show selected shape only function Expand and collapsing a shape: unhide the links and nodes used by the affected shape's constraints On selecting a shape and this toggle is on: some shapes are partially hidden and should re-show their constraints.

5.3.7 Light and Dark Mode

Changing from dark theme to light theme was done by presenting two lists of CSS variables for colors, those variables are used instead of writing colors directly in the CSS code. Now, when the user changes the mode, a new attribute is set for the whole page called `data-theme` and its values are *light* or *dark*. This will change the color variable list for the page and the graph, but an update for the graph is required to get the effect Figure 5.9.



(a) Intra-Constraints not hidden (b) Intra-Constraints hidden

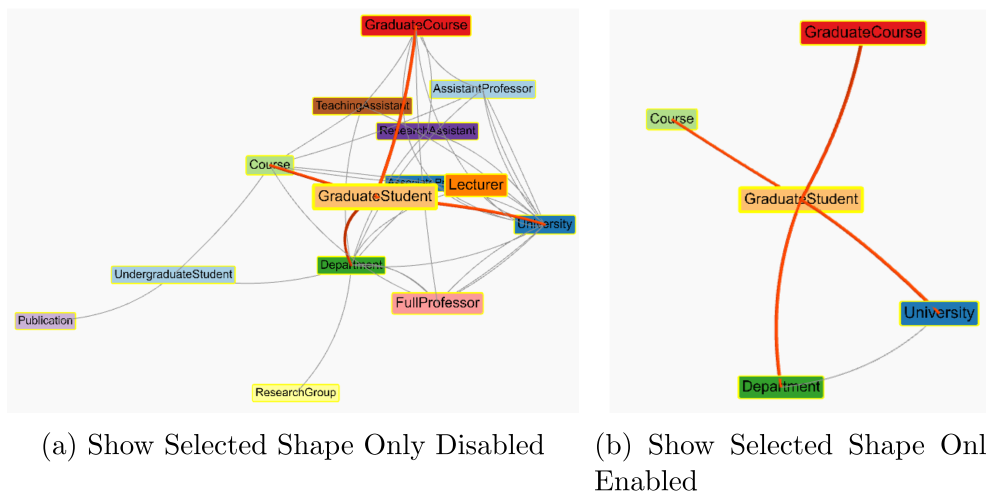
Figure 5.7: **Hide Intra-Constraints** By enabling this feature all intra-constraints will be hidden.

5.3.8 Info Panel

In this panel there is two parts, meta data and the constraints. The meta data numbers are the saved in the statistics variable in each shape and they are: intra-constraints, inter-constraints and number of constraints targeting this node. In the second part, the constraints tree for the selected shape, the JavaScript tree view is simple-treeview [8] (version 0.0.9). The head of the tree is the selected shape, followed by the intra-constraints then the inter-constraints. Clicking on one of the constraints will go deeper in the tree and show what are the conditions of that constraint, and if the constraint is an inter-constraint the targeted shape will be presented. Clicking on the shape from the tree view will expand the shape in the graph without making that shape the selected shape, and the tree view will go deeper showing all its constraints as it did for the first selected shape Figure 5.10.

5.3.9 Shape Checklist Panel

This check list Figure 5.11 is filled while loading the page with shape nodes. It is done by adding an event on each one of them where, if clicked the node with the same ID, will have its `is hidden` property flipped with their constraint nodes. Then, an update for hidden links function is called to check links with hidden source node or target node to be hidden too. Then, the graph clicked list are updated. The selected node is only used, and the shape checked list panel is open.



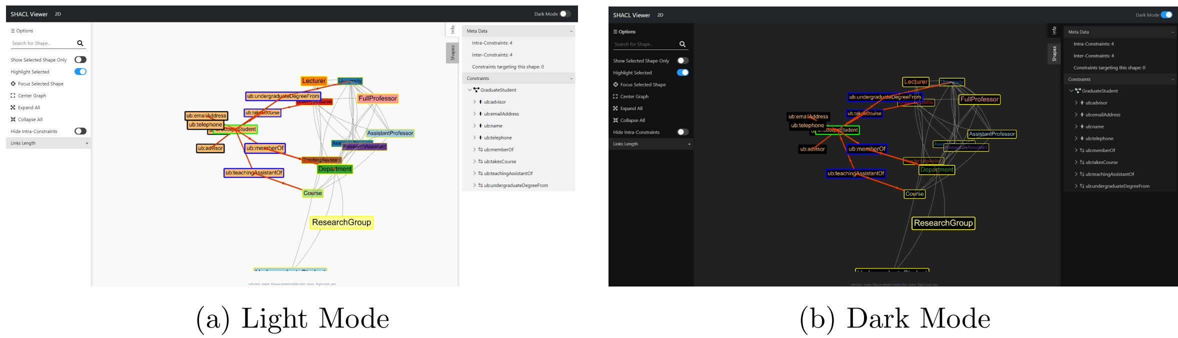
(a) Show Selected Shape Only Disabled (b) Show Selected Shape Only Enabled

Figure 5.8: **Show Selected Shape Only** By enabling this feature all shapes that are not connected by the selected shape constraints are hidden.

5.3.10 *force-graph* Supported Functions

The remaining features are supported by the *force-graph* libraries [3, 4] and they can be used from the options menu Figure 5.12 and they are:

- Focus selected shape: uses a function to focus a node, in this case it is the selected shape node
- Center graph: a function to put the average nodes' position on the zero axes of the 3D space.
- Group coloring: this feature will give a random color for each node group; it is used to give each shape nodes a different color.
- Link curve: used for shape-to-shape links only to distinguish them from other link types.
- Link length: used to update the value of each slider in the option menu, it will change the link length for each link type.
- Navigation: panning, rotation and zooming are used to see the graph from different angles in the 3D or 2D space.
- Drag and drop: the ability to move the nodes around while maintaining a good distance from each other by defining d3Force value.



(a) Light Mode

(b) Dark Mode

Figure 5.9: **Light and Dark Mode** By enabling this feature all colors will change.

5.4 Summary of the Chapter

This chapter presented the *SHACLViewer* implementation. The data transformation from JSON files to a list of shapes, then how it was converted to nodes and links. Also, how each feature is done and seen what the user expects when using them.

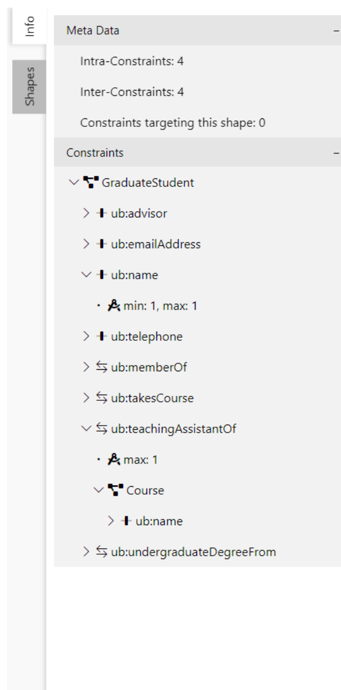


Figure 5.10: **Info Panel** This panel shows information about the selected shape and a tree view for its constraints.

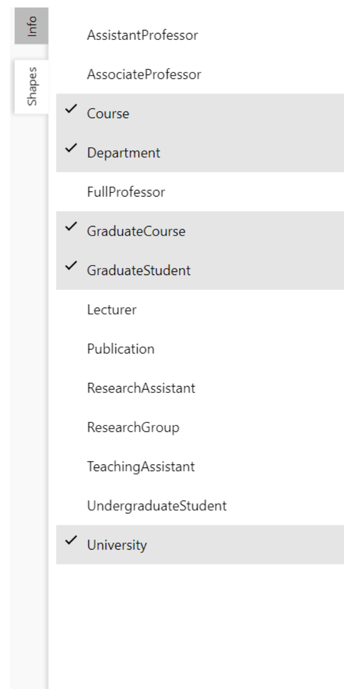


Figure 5.11: **Shape Checklist Panel** This panel shows all shapes listed in a check list where the user could show or hide any shape.

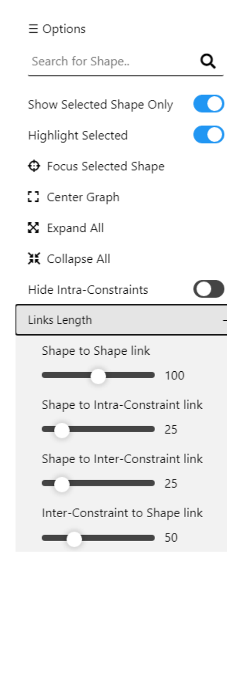


Figure 5.12: **Options Menu** This menu contains most of the features supported by *SHACLViewer*.

Chapter 6

Experimental Evaluation

We performed an experimental evaluation of the scalability of *SHACLViewer*. The evaluation was designed to address the following research questions: **RQ1)** What is the impact of the size of the SHACL shape schema? **RQ2)** What is the impact of the topology of the SHACL shape schema? **RQ3)** What is the impact of the density of the SHACL shape schema? The source code of *SHACLViewer* is available at GitHub[2]. Next, the experimental setup as well as the observed results are presented.

6.1 Experimental Setup

This section describes the setup of the experimental evaluation conducted in order to answer the above-mentioned research questions. In the following, the benchmarks, metrics, and the general setup are explained in detail.

6.1.1 Benchmarks

The performance of *SHACLViewer* is evaluated over synthetic and real-world SHACL shape schemas. The real-world shape schemas represent sets of constraints one expects in real-world scenarios. The goal is to show that *SHACLViewer* is able to handle SHACL shape schemas of expected complexities. The synthetic shape schemas are a stress test for the scalability of *SHACLViewer*. In the following, both sets of SHACL shape schemas are described in more detail.

[2] https://github.com/SDM-TIB/SHACLViewer

Synthetic Benchmarks

The synthetic benchmarks serve as a stress test for the scalability of *SHACLViewer*. In order to generate synthetic benchmarks, we use `networkx`[3] to generate different networks. Each node in the network is considered to be a SHACL shape while the edges represent inter-constraints. For simplicity, we do not add intra-constraints to the synthetic shapes. Following this procedure, we create SHACL shapes schemas for nine different topologies. For each topology, SHACL shape schemas with around 25, 50, 75, 100, 150, and 200 shapes are generated. This leads to a total of 54 synthetic benchmarks. In the following, the topologies are described. Different graph measures as well as an example graph for each topology can be found in Table 6.1.

- Binary Tree: A tree graph where each node in the graph has at most two successors, i.e., the tree is not balanced.
- Circulant Graph: A circulant graph $Ci_n(x_1, x_2, \dots, x_m)$ consists of n nodes such that each node i is connected to nodes $(i + x) \mod n$ and $(i - x) \mod n$ for all $xinx_1, \dots, x_m$. Hence, $Ci_n(1)$ is a cycle graph. We choose $x_1 = 5$, $x_2 = 10$.
- Complete Graph: In a complete graph, all nodes are connected to each other.
- Cycle Graph: A cycle graph C_n is a graph of cyclically connected nodes. C_n is a path with its two end-nodes connected.
- Ladder Graph: Two paths of length n with each pair connected by a single edge.
- Path Graph: A path graph consists of linearly connected nodes.
- Star Graph: A star graph consists of one center node connected to n outer nodes.
- Turan Graph: A complete multipartite graph of n nodes with r disjoint subsets, i.e., edges connect each node to every node not in its subset. We choose $r = 3$.
- Wheel Graph: A graph with one hub node connected to a cycle of $n - 1$ nodes.

Complete graphs and Turan graphs have a high connectivity, resulting in a high density. Due to the high density, the complexity of the resulting SHACL shape schema is higher for those topologies compared to the others.

[3] https://networkx.org/

Table 6.1: **Synthetic SHACL Shape Schemas.** Nine different topologies of six different sizes. #shapes - number of shapes; #con - number of constraints; density - density; #nodes - number of nodes needed; #links - number of links

Topology 1: Binary Tree

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	24	49	74	99	149	199
density	0.080	0.040	0.027	0.020	0.013	0.010
#nodes	73	148	223	298	448	598
#links	144	294	444	594	894	1,194

Topology 2: Circulant Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	50	100	150	200	300	400
density	0.167	0.082	0.054	0.040	0.027	0.020
#nodes	125	250	375	500	7500	1,000
#links	300	600	900	1,200	1,800	2,400

Topology 3: Complete Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	600	2,450	5,550	9,900	22,350	39,800
density	1.000	1.000	1.000	1.000	1.000	1.000
#nodes	625	2,500	5,625	10,000	22,500	40,000
#links	1,800	7,350	16,650	29,700	67,050	119,400

Topology 4: Cycle Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	25	50	75	100	150	200
density	0.083	0.041	0.027	0.020	0.013	0.010
#nodes	75	150	225	300	450	600
#links	150	300	450	600	900	1,200

Topology 5: Ladder Graph

	1	2	3	4	5	6
#shapes	24	50	74	100	150	200
#con	34	73	109	148	223	298
density	0.123	0.060	0.040	0.030	0.020	0.015
#nodes	92	196	292	396	596	796
#links	204	438	654	888	1,338	1,788

Topology 6: Path Graph

	1	2	3	4	5	6
#nodes	25	50	75	100	150	200
#edges	24	49	74	99	149	199
density	0.080	0.040	0.027	0.020	0.013	0.010
#nodes	73	148	223	298	448	598
#links	144	294	444	594	894	1,194

Topology 7: Star Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	24	49	74	99	149	199
density	0.080	0.040	0.027	0.020	0.013	0.010
#nodes	73	148	223	298	448	598
#links	144	294	444	594	894	1,194

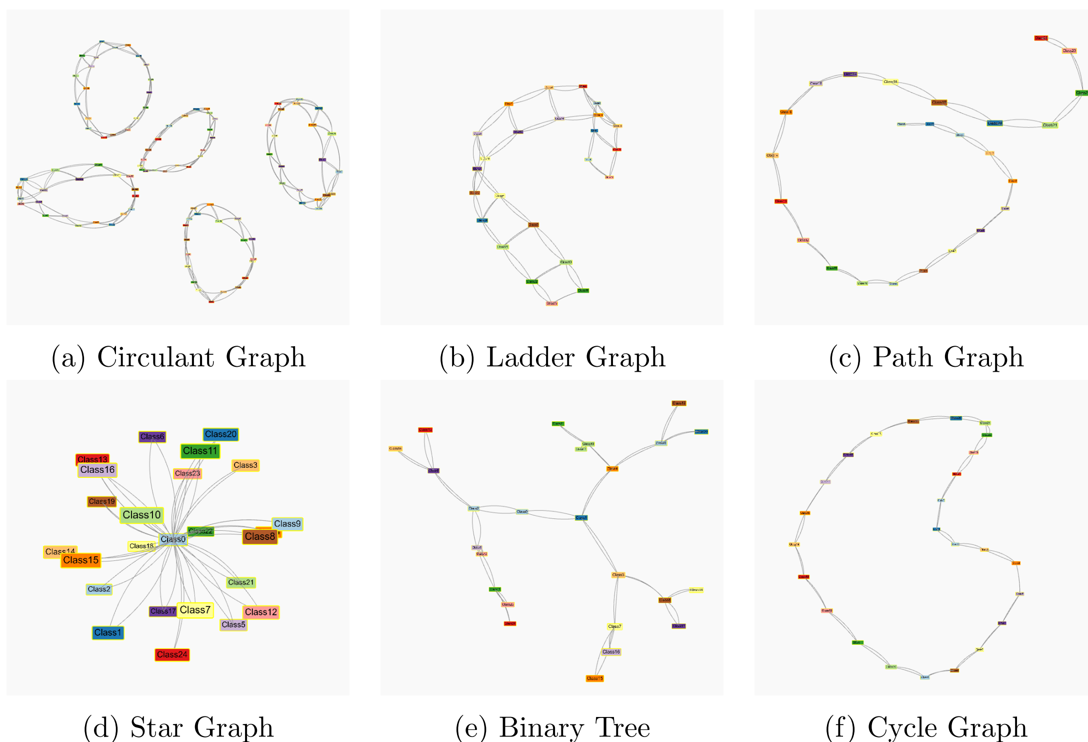
Topology 8: Turan Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	208	833	1,875	3,333	7,500	13,333
density	0.693	0.680	0.676	0.673	0.671	0.670
#nodes	144	1,716	3,825	6,766	15,150	26,866
#links	1,248	4,998	11,250	19,998	45,000	79,998

Topology 9: Wheel Graph

	1	2	3	4	5	6
#shapes	25	50	75	100	150	200
#con	48	98	148	198	298	398
density	0.160	0.080	0.053	0.040	0.027	0.020
#nodes	121	246	371	496	746	996
#links	288	588	888	1,188	1,788	2,388

Figure 6.1 depicts a 3D view rendering generated powered by *SHACLViewer*.



(a) Circulant Graph (b) Ladder Graph (c) Path Graph

(d) Star Graph (e) Binary Tree (f) Cycle Graph

Figure 6.1: *SHACLViewer* Visualization of Synthetic Graphs with 25 shapes.

Real-World Benchmarks

To show that *SHACLViewer* can also handle real-world SHACL shape schemas, we use five shape schemas over well-known benchmarks (LUBM [15], WatDiv [1]) and DBPedia [5, 20]. The shape schema over DBPedia consists of nine shapes over classes like `dbo:Person`, `dbo:Film`, `dbo:Disease`. The integrity constraints represented in the shape schema are inspired by the quality assessment study presented by Kontokostas et al. [18]. The shape schema *LUBM_14* is the same as used by Figuera et al. in the Trav-SHACL [14] experiments [13]. *LUBM_5* is a subset of *LUBM_14* only considering the shapes for departments, full professors, graduate courses, graduate students, and universities. *LUBM_23* was created from automatically extracted SHACL shapes published by Rabbani et al. [24]. We download the extracted shape schema and keep the minimal cardinality for each predicate. The approach used by Rabbani et al. does not generate references to other shapes, hence, the shapes of *LUBM_23* are not connected. For WatDiv we create shapes for the super-classes `Genre`, `ProductCategory`, `Role`. The minimal and maximal cardinalities were set in a manner that ensures valid as well as invalid instances. However, the validation of SHACL constraints is out of the scope of this thesis. Common graph measures for the real-world shape schemas are presented in Table 6.2.

Table 6.2: **Statistics of Real-World SHACL Shape Schemas.** The real-world shape schemas are expressed over three different data sets, namely *DBPedia*, *LUBM*, and *WatDiv*. #shapes - number of shapes in the shape schema, #con - number of constraints in the shape schema, density - density of the shape schema

Shape Schema	#shapes	#con	density
DBPedia	9	31	0.194
LUBM_5	5	23	0.600
LUBM_14	14	110	0.308
LUBM_23	23	136	0.000
WatDiv	3	58	0.667

6.1.2 Metrics

We report the following metrics:

- *Load Time:* Time elapsed between choosing a SHACL shape schema to visualize until it was fully loaded.
- *Expand Time:* Time elapsed between requesting the entire SHACL shape schema to expand and the completion of the expanding action.
- *Collapse Time:* Time elapsed between requesting the SHACL shape schema to collapse and the completion of the collapsing action.
- *Highlight Expanded Time:* Time elapsed between requesting a node to be highlighted and the node reaching the highlighted state while the SHACL shape schema is expanded.
- *Highlight Collapsed Time:* Time elapsed between requesting a node to be highlighted and the node reaching the highlighted state while the SHACL shape schema is collapsed.

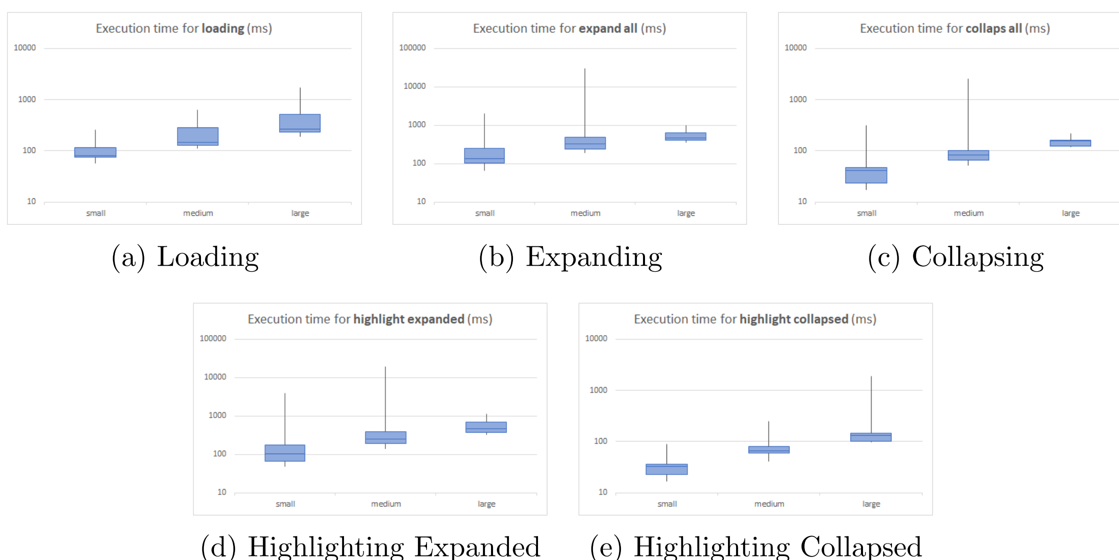
All times measured correspond to absolute wall-clock system time in milliseconds.

6.1.3 Setup

The above-mentioned metrics are collected for each of the 59 benchmarks, i.e., the 54 synthetic and the five real-world SHACL shape schemas. The Flask Development Server was used while conducting the experiments. Furthermore, the Flask application was configured to not cache results. The experiments are executed on a Windows 10 Pro 64 bit machine with an Intel® Core™ i7-8750H CPU, and 16 GiB

RAM. The graphics card is an NVIDIA GeForce GTX 1070 with 8 GiB VRAM. The Google Chrome version used is 99.0.4844.74; hardware acceleration is enabled.

6.2 Impact of Shape Schema Size



(a) Loading (b) Expanding (c) Collapsing

(d) Highlighting Expanded (e) Highlighting Collapsed

Figure 6.2: **Size Impact on Execution Time (ms).** All functions are impacted by the size, i.e., number of shapes, of the shape schema. Expanding seems to be the least impacted, this is due to the fact that the large complete graph and Turan graph caused *SHACLViewer* to crash, hence, no times are reported.

For the first test the size of the networks is combined in three separate categories; small, medium and large, where small are networks with shape count of less than 75, medium networks are from 75 to 100 shapes, and large are networks with more than 100 shapes. Figure 6.2 reveals that all functions are impacted by increasing the number of shapes in a network. Some networks resulted in SHACLViewer crashing mainly *Complete* and *Turan* schemas large graphs of 150 and 200 shapes, when performing the function *Expand all* shapes and that prevented the test of *Highlighting expanded* shapes and *Collapse all* shapes. And it is visible in the results for large graphs in *Expanding* Figure 6.2b, *Collapsing* Figure 6.2c and *Highlight Expanded* Figure 6.2d.

Beside the number of shapes, in *SHACLViewer* each shape consists of a different number of nodes and links depending on how many constraints a shape contains. which can be calculated using these equations: $Nodes = Shapes + Constraints$ and

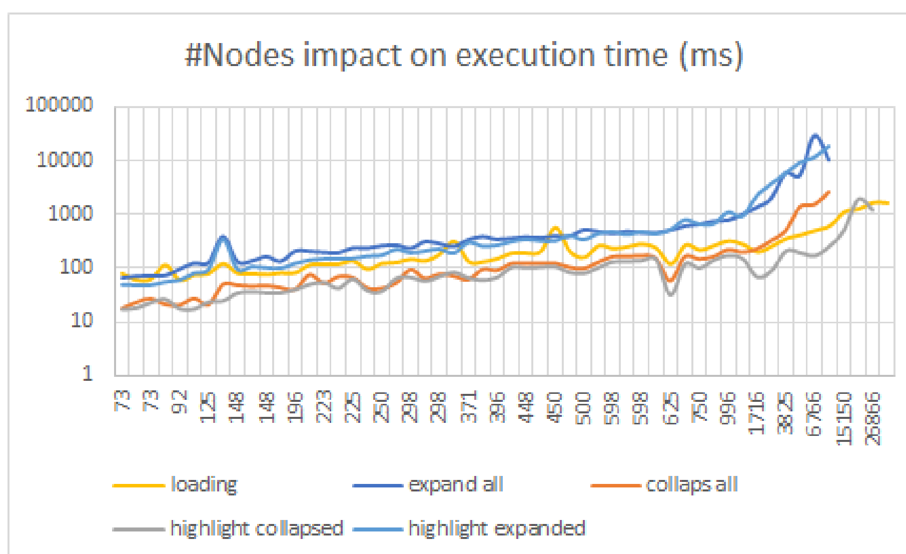


Figure 6.3: **Impact of Number of Nodes on Execution Time (ms).** All functions are impacted by the number of nodes in the graph visualization. Expanding and highlighting (expanded) are impacted the most. Loading and highlighting (collapsed) are the least affected. At about 2,000 nodes, the cost increases dramatically.

$Links = intra\text{-}constraints + 3 * inter\text{-}constraint$. In Table 6.2 the statistics for each network and how may nodes and links are required to represent it. That is mostly the reason why *Complete* and *Turan* schemas with 150 and 200 shapes crashed, while other schemas did not. On that note, the impact of number of nodes and links is visualized in Figure 6.3 and Figure 6.4 respectively; as expected, the execution time dramatically increases at about 2000 nodes or 2500 links.

6.3 Impact of Shape Schema Topology

The topology of the network will change the time spent executing the functions for each network as shown in Figure 6.5, this change is due to the number of constraints each topology presents, and it can be seen from *Complete* and *Turan* topologies statistics (see Table 6.1) that they are the most expensive ones.

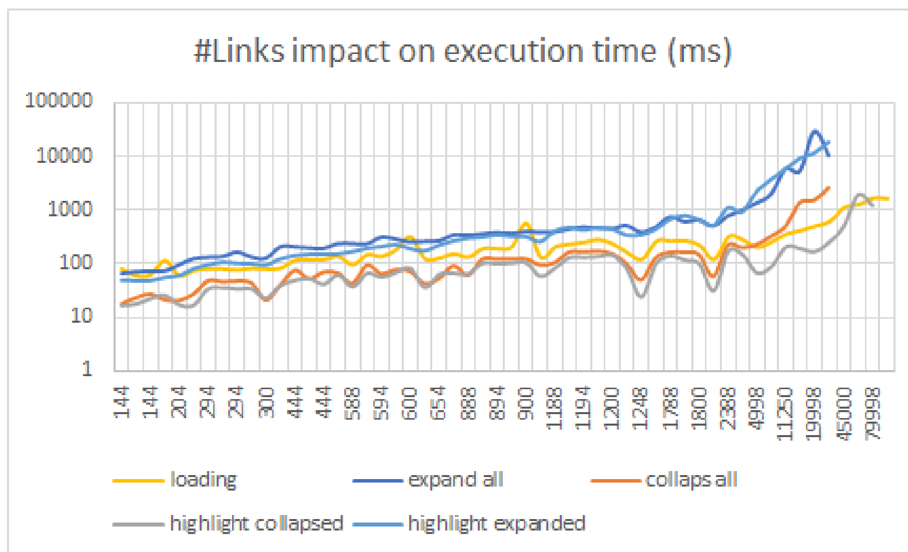


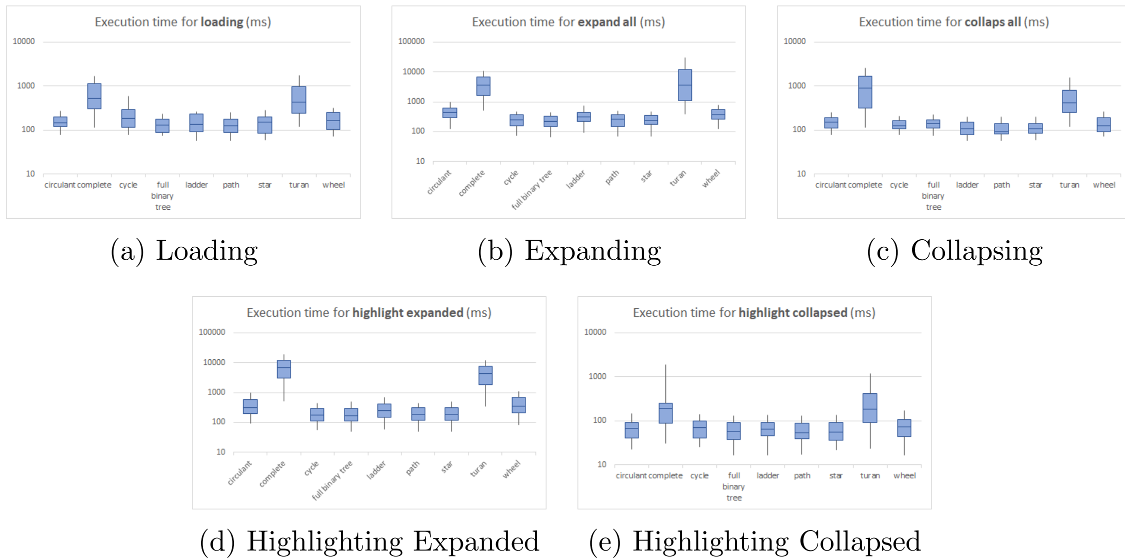
Figure 6.4: **Impact of Number of Links on Execution Time (ms).** All functions are impacted by the number of links in the graph visualization. Expanding and highlighting (expanded) are impacted the most. Loading and highlighting (collapsed) are the least affected. At about 2,500 links, the cost increases dramatically.

6.4 Impact of Shape Schema Density

In Figure 6.6 it is visible the execution time did not change dramatically until it reaches the density of 0.670, and that alone will not be enough to get a conclusion. In Figure 6.7 the number of nodes and links is added to Figure 6.6, now we can see that the number of nodes and links is matching the spikes and irregularities in the execution times, meaning it has more effect on the execution time than density.

6.5 Real-World Shape Schemas

For real-world SHACL shape network schemas, number of nodes and links are also calculated as shown in Figure 6.8. Then the values for each metric function are presented in Figure 6.9. *LUBM_23* has the most nodes while *LUBM_14* has the most links, but the execution times for *LUBM_23* are higher in general than *LUBM_14*, meaning the number of nodes is impacts most on the execution times.



(a) Loading (b) Expanding (c) Collapsing

(d) Highlighting Expanded (e) Highlighting Collapsed

Figure 6.5: **Topology Impact on Execution Time (ms).** It is clear that all functions are impacted by the topology of the SHACL shape schema. As expected, complete graphs and Turan graphs are the most expensive topologies.

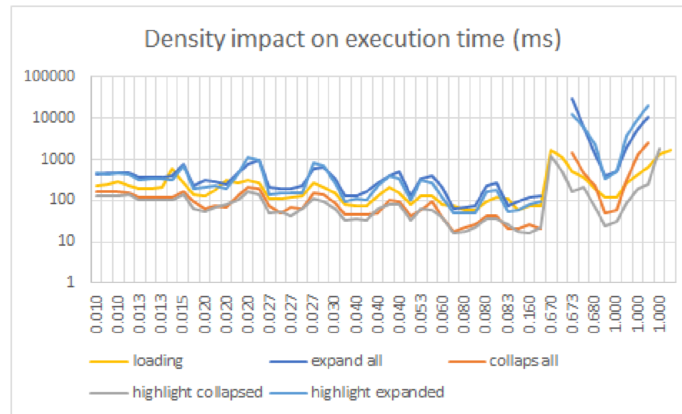


Figure 6.6: **Density Impact on Execution Time (ms).** The diagram reports the execution time (ms) of the different metrics as a function of the SHACL shape schema density. Expanding and highlighting in the expanded graph are impacted the most by high density. Collapsing the graph is the least affected by density.

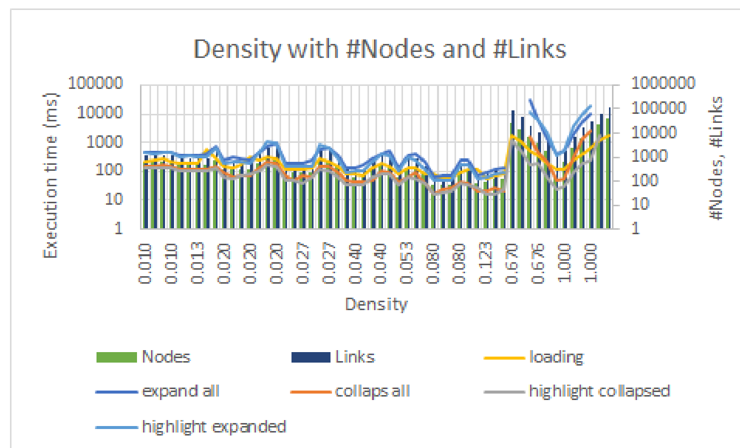


Figure 6.7: **Density Impact on Execution Time (ms) with number of Nodes and Links** The diagram reports the execution time (ms) of the different metrics as a function of the SHACL shape schema density, with number of nodes and links for each value. It is visible that the numbers for nodes and links have a higher impact on the execution time than density.

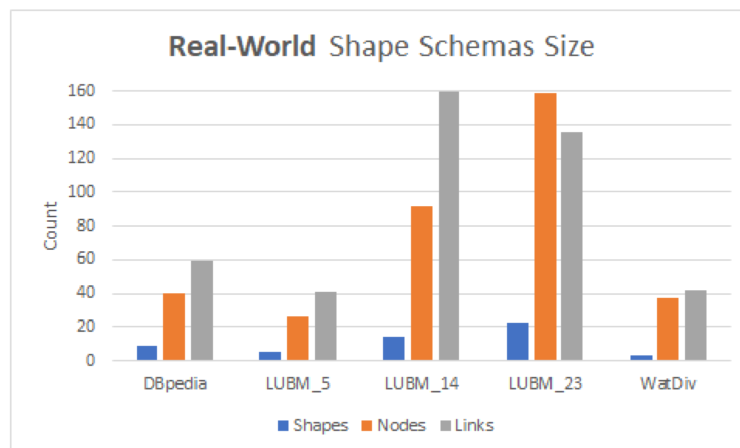


Figure 6.8: **Size of Real-World Shape Schemas.** The diagram reports the number of shapes, nodes and links for each real-world network. *LUBM_23* has the most number of nodes, while *LUBM_14* has the most number of links; this will help deciding which metric has the highest impact on the execution time.

6.6 Summary of the Chapter

For the experimental evaluation, multiple synthetic SHACL shape schemas of different sizes and topologies were created. Five metrics were evaluated to analyze the

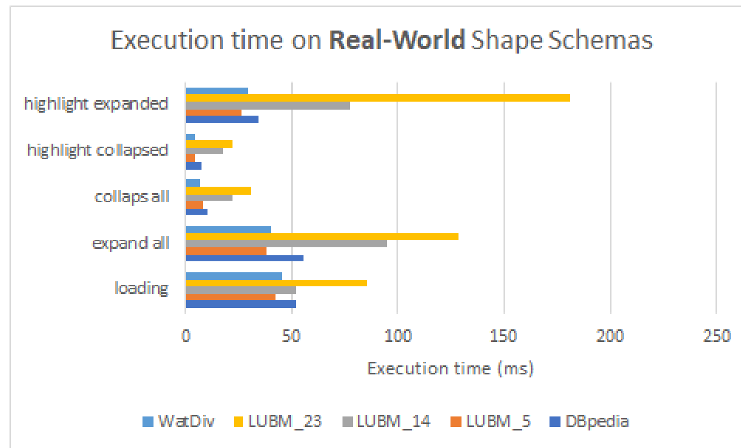


Figure 6.9: **Execution Time (ms) on Real-World Shape Schemas.** Execution time (ms) of the different metrics for each of the real-world shape schemas. *LUBM_14* and *LUBM_23* are the most expensive SHACL shape schemas. The most expensive functions are expanding and highlighting in the expanded graph.

scalability of *SHACLViewer*. Next, five real-world shape schemas were tested. To conclude, the most expensive functions are *Expand All* and *Highlight while expanded*, and the toughest topologies to work with are *Complete* and *Turan*; and that is because the number of nodes needed to present a network has a higher impact than the number of links.

Chapter 7

Conclusions and Future Work

This thesis presented a new way to view SHACL networks by drawing it in 3D rather in 2D, then collapsing all shapes into one node connected with one simple link for each constraint it has, making it easier for the user to check all the network without too many details. Then by clicking on a shape's node it will expand showing all the details for its constraints. Also, there is the ability to hide all shapes but one, then the user could add other shapes as necessary.

7.1 Conclusions

There are many ways to visualize SHACL networks and 3D is one of them, the most important part when visualizing a complex graph is the features available to the user, those features that will simplify navigating the graph, showing what the user is trying to find and hiding any unrelated data while giving the user full control. All that while maintaining a simple, easy to use user interface that is quickly understandable. The experimental evaluation of the proposed approach was aimed at addressing what has the most impact on the performance. For **RQ1**, the size of the network was discussed, and there were two types of size to discuss, first shapes count in the network, which had a significant impact on the performance. the second was the number of nodes and links required to render the network, which had the most impact. Next in **RQ2**, the focus was on the topology of the SCHACL shape schema, where *Complete* and *Turan* had a bigger impact than other topologies. and last for **RQ3** the density of the schema was tackled, the density alone did not give an accurate representation for the time taken to perform the tasks. So, the conclusion was the most impactful metric is the number of nodes and links required to render it.

7.2 Limitations

As reported in Chapter 6, SHACLViewer's scalability is impacted by the SHACL network complexity. Specifically, the parameters that negatively impact the performance of SHACLViewer include; a) size (i.e., more than 2,500 nodes); b) density (i.e., highly dense graphs); and c) graph topology (i.e., complete and Turan graphs). Moreover, SHACLViewer crashes in networks with 5,000 nodes.

7.3 Future Work

Addressing the limitations of the current version of SHACLViewer demand redesign of the tool's functions, i.e., loading, expand, collapse, and highlight. Extensions to be considered as future work include:
Compact representation of shapes. A new way to reduce the number of nodes in a single graph by merging multiple shapes into a single node. It will look like each node is a cluster of SHACL shapes.
Compact representation of shape connections. In the current form of "show selected shape only", the selected shape will be shown with its neighboring shapes connected with constraints. A new feature could be added to show an N-degree neighborhood for the selected node and hide the rest.
Fine-Grained Highlight. Also, another toggle could be added to show any other shapes with inter-constraints targeting the selected shape highlighted with different color and shown when "selected shape only" is used.
Dimension quick swap. Currently, changing from 3D to 2D will reload the page losing the selected shape and any customization like hidden or expanded shapes. A better way is to change the graph container directly saving any changes to Nodes and Links parameters.
Online SHACL Network Validation. Another feature is adding an API to highlight specific shapes or constraints from other applications or services like a SHACL validator, using SHACLViewer as an advanced tool to view a validated network.
Searching shapes. Add a search box to the shape checklist to easily find shapes.

7.4 Summary of the Chapter

This chapter discussed *SHACLViewer*'s experimental results as well as the cases in which the limitations were presented. Future works that can be added for a feature-rich application were also suggested.

Bibliography

[1] Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. "Diversified Stress Testing of RDF Data Management Systems". In: *The Semantic Web – ISWC 2014*. Cham: Springer, 2014, pp. 197–212. DOI: 10.1007/978-3-319-11964-9_13.

[2] Grigoris Antoniou and Frank van Harmelen. "Web ontology language: Owl". In: *Handbook on ontologies*. Springer, 2004, pp. 67–92.

[3] Vasco Asturiano. *3d-force-graph*. JavaScript Library hosted on GitHub. 2017. URL: https://github.com/vasturiano/3d-force-graph.

[4] Vasco Asturiano. *force-graph*. JavaScript Library hosted on GitHub. 2018. URL: https://github.com/vasturiano/force-graph.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. "DBpedia: A Nucleus for a Web of Open Data". In: *The Semantic Web*. Ed. by Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 722–735. ISBN: 978-3-540-76298-0.

[6] L. Bassett. *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. O'Reilly Media, 2015. ISBN: 9781491929438. URL: https://books.google.de/books?id=Z%5C_9PCgAAQBAJ.

[7] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web*. Scientific American. May 2001. URL: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[8] Petr Broz. *simple-treeview*. JavaScript Library hosted on GitHub. 2021. URL: https://github.com/petrbroz/simple-treeview.

[9] Julien Corman, Fernando Florenzano, Juan L. Reutter, and Ognjen Savković. "Validating SHACL Constraints over a SPARQL Endpoint". In: *The Semantic Web – ISWC 2019*. Cham: Springer, 2019, pp. 145–163. DOI: 10.1007/978-3-030-30793-6_9.

[10] D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. Request for Comments: 4627. July 2006. URL: https://www.ietf.org/rfc/rfc4627.txt.

[11] Marek Dudáš, Steffen Lohmann, Vojtěch Svátek, and Dmitry Pavlov. "Ontology visualization methods and tools: a survey of the state of the art". In: *The Knowledge Engineering Review* 33 (2018). DOI: 10.1017/S0269888918000073.

[12] Riccardo Falco, Aldo Gangemi, Silvio Peroni, David Shotton, and Fabio Vitali. "Modelling OWL Ontologies with Graffoo". In: *The Semantic Web: ESWC 2014 Satellite Events*. Ed. by Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai. Cham: Springer International Publishing, 2014, pp. 320–325. ISBN: 978-3-319-11955-7. URL: https://2014.eswc-conferences.org/sites/default/files/eswc2014pd_submission_114.pdf.

[13] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. *Dataset: Trav-SHACL: Benchmarks, Experimental Settings, and Evaluation*. Leibniz University of Hannover Data Repository. Feb. 2021. DOI: 10.25835/0035739.

[14] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. "Trav-SHACL: Efficiently Validating Networks of SHACL Constraints". In: *The Web Conference*. New York, NY, USA: ACM, 2021, pp. 3337–3348. DOI: 10.1145/3442381.3449877.

[15] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A Benchmark for OWL Knowledge Base Systems". In: *Web Semantics* 3.2–3 (2005), pp. 158–182. DOI: 10.1016/j.websem.2005.06.005.

[16] Claudio Gutiérrez and Juan F Sequeda. "Knowledge graphs". In: *Communications of the ACM* 64.3 (2021), pp. 96–104.

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

[18] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. "Test-Driven Evaluation of Linked Data Quality". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. New York, NY, USA: ACM, 2014, pp. 747–758. DOI: 10.1145/2566486.2568002.

[19] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. Feb. 1999. URL: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[20] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. "DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195. DOI: 10.3233/SW-140134. URL: http://svn.aksw.org/papers/2013/SWJ_DBpedia/public.pdf.

[21] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. "Visualizing ontologies with VOWL". In: *Semantic Web* 7.4 (2016), pp. 399–419. DOI: 10.3233/SW-150200. URL: http://www.semantic-web-journal.net/system/files/swj1114.pdf.

[22] Rossana Pacielloa, Daniele Bailoa, Luca Tranib, Valerio Vinciarellia, Manuela Sbarrac, and Sara Capotostic. *SHAPEness: a SHACL-driven RDF Graph Editor*. Rejected submission to the Semantic Web Journal. 2021. URL: http://www.semantic-web-journal.net/content/shapeness-shacl-driven-rdf-graph-editor.

[23] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation. Feb. 2004. URL: https://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[24] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. *Automatically Extracted SHACL Shapes for DBpedia, YAGO-4, and LUBM & Associated Coverage Statistics*. zenodo. Feb. 2022. DOI: 10.5281/zenodo.5958986.

[25] Maria-Esther Vidal, Kemele M. Endris, Samaneh Jazashoori, Ahmad Sakor, and Ariam Rivas. "Transforming Heterogeneous Data into Knowledge for Personalized Treatments - A Use Case". In: *Datenbank-Spektrum* 19.2 (2019), pp. 95–106.

[26] Vitalis Wiens, Steffen Lohmann, and Sören Auer. "GizMO – A Customizable Representation Model for Graph-Based Visualizations of Ontologies". In: *Proceedings of the 10th International Conference on Knowledge Capture*. K-CAP '19. Marina Del Rey, CA, USA: Association for Computing Machinery, 2019, pp. 163–170. ISBN: 9781450370080. DOI: 10.1145/3360901.3364431. URL: https://doi.org/10.1145/3360901.3364431.