Visual Exploration of Semantic-Web-Based Knowledge Structures

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades

Doktor-Ingenieur
(abgekürzt Dr.-Ing.)
genehmigte Dissertation

von Herrn
M.Sc. Vitalis Wiens

2022

1. Referent	Prof. Dr. Sören Auer, Leibniz Universität Hannover
2. Referent	Prof. Dr. Dmitry Muromtsev, ITMO University (RU)
3. Referent	Prof. Dr. Ziawasch Abedjan, Leibniz Universität Hannover
Tag der Promotion	10.05.2022

Zusammenfassung

Daten, Informationen, und Wissen wurden durch die informationstechnische Revolution, in der Form des Internets, zu Gütern unserer modernen Gesellschaft. Jedoch, mit der wachsenden Größe der angesammelten Daten entstehen neue Herausforderungen, wie z.B. das Suchen und Navigieren in diesen großen Ansammlungen von Daten, Informationen, und Wissen. In akademischen und industriellen Kontexten werden diese Herausforderungen mit Semantic Web Technologien adressiert. Das Semantic Web ist eine Erweiterung des Webs und stellt maschinenlesbare Repräsentationen von Wissen für verschiedene Domänen dar. Außerdem, erlaubt das Semantic Web maschinellen Agenten Daten und Informationen zu verstehen und ermöglichen zusätzlich die Inferenz von neuem Wissen.

Das Semantic Web ist für den Austausch von Informationen und deren Verarbeitung konzipiert und konzentriert sich nicht auf die Präsentation von semantischen Daten für den Menschen. Visualisierungen unterstützen die Erkundung, Navigation und das Verständnis von Daten, indem sie die Fähigkeit des Menschen ausnutzen komplexe Daten durch visuelle Darstellungen zu verstehen. Geeignete Visualisierungen sind jedoch stark abhängig von den individuellen Anwendungsfällen und deren Nutzergruppen.

In dieser Arbeit untersuchen wir visuelle Explorationstechniken für Semantic Web Daten, indem wir die folgenden Herausforderungen adressieren: **i)** wie kann man verschiedene Benutzergruppen in die Ontologiemodellierung einbinden, **ii)** wie kann man das Verständnis durch anpassbare visuelle Darstellungen erleichtern und **iii)** wie kann man die Erstellung von Visualisierungen für verschiedene Anwendungsfälle erleichtern. Die erzielten Ergebnisse zeigen auf, dass visuelle Modellierungstechniken verschiedene Benutzergruppen bei der Ontologiemodellierung unterstützen. Anpassbare Visualisierungen ermöglichen es den Benutzern, die Darstellung an ihre aktuellen Bedürfnisse anzupassen und unterschiedliche Sichten auf die Daten zu erstellen. Zusätzlich ermöglichen anpassbare Visualisierungspipelines die schnelle Erstellung von Visualisierungen für verschiedene Anwendungsfälle, Datenquellen, und Benutzergruppen.

Schlagwörter:

Semantic Web,

Anpassbare Visuelle Darstellungsmodelle,

Ontologievisualisierung,

Visuelle Modellierungstechniken,

Visualisierungspipelines

Abstract

Humans have a curious nature and seek a better understanding of the world. Data, information, and knowledge became assets of our modern society through the information technology revolution in the form of the internet. However, with the growing size of accumulated data, new challenges emerge, such as searching and navigating in these large collections of data, information, and knowledge. The current developments in academic and industrial contexts target the corresponding challenges using Semantic Web technologies. The Semantic Web is an extension of the Web and provides machine-readable representations of knowledge for various domains. These machine-readable representations allow intelligent machine agents to understand the meaning of the data and information; and enable additional inference of new knowledge.

Generally, the Semantic Web is designed for information exchange and its processing and does not focus on presenting such semantically enriched data to humans. Visualizations support exploration, navigation, and understanding of data by exploiting humans' ability to comprehend complex data through visual representations. In the context of Semantic-Web-Based knowledge structures, various visualization methods and tools are available, and new ones are being developed every year. However, suitable visualizations are highly dependent on individual use cases and targeted user groups.

In this thesis, we investigate visual exploration techniques for Semantic-Web-Based knowledge structures by addressing the following challenges: **i)** how to engage various user groups in modeling such semantic representations; **ii)** how to facilitate understanding using customizable visual representations; and **iii)** how to ease the creation of visualizations for various data sources and different use cases. The achieved results indicate that visual modeling techniques facilitate the engagement of various user groups in ontology modeling. Customizable visualizations enable users to adjust visualizations to the current needs and provide different views on the data. Additionally, customizable visualization pipelines enable rapid visualization generation for various use cases, data sources, and user groups.

Keywords:

Semantic Web,

Customizable Visual Representation Models,

Ontology Visualization,

Visual Modeling,

Visualization Pipelines

Acknowledgments

Throughout the exciting academic journey, I met various people who supported and inspired me in all those years. I would like to thank Prof. Dr. Sören Auer for giving me an opportunity to pursue my PhD degree at Fraunhofer, TIB, and Leibniz University of Hannover. Furthermore, I would like to thank Prof. Dr. Maria-Esther Vidal for her invaluable scientific writing mantra: “What?, Why?, and How?” Especially, I would like to thank Dr. Steffen Lohmann for his help, inspiration, advice, and fruitful scientific discussions. Additionally, I would like to thank Dr. Markus Stocker for his supervision during my time at the Leibniz University of Hannover.

Special thanks goes to Micheal Ankele, Debanjan Chaudhuri, Diego Collarana, Kheir Eddine Farfar, Michael Galkin, Damien Graux, Max Herrmann, Julian Iseringhausen, Yaser Jaradeh, Samaneh Jozashoori, Rostislav (Rosko) Nedelchev, Allard Oelen, Simon Scerri, Marc Schultheis, Micheal Weinmann, Manuel Widdel, and Yun for their support and encouragement.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The need to explore and understand a rapidly changing world reflects George Miller's [1] characterization of the human species as *informavores*. Humans have a curious nature and are hungry for information, seeking a better understanding of the world [2]. Libraries have long pursued the vision of accumulated human knowledge by collecting books and providing access to them. However, the information in such collections is challenging to share because it requires a physical access. Additionally, the knowledge is buried in textual descriptions, making searching, organizing, and understanding time-consuming efforts.

In the form of the internet, the information technology revolution simplified collecting and exchanging data. Thus, we evolved towards an information society that hunts for and gathers information. As a result, data and information became assets, and we frequently hear the term *data is the new oil*. However, with the increasing size of accumulated data, it became challenging to search, explore, and navigate these massive collections. Thus, search engines and online encyclopedias such as Google and Wikipedia emerged to facilitate searching and gathering of information. Nevertheless, searching and organizing information in textual representations, e.g., in the form of articles, remain cumbersome.

The current developments in industrial and academic contexts address challenges in knowledge management using Semantic Web technologies. These technologies provide machine-readable representations of domain knowledge, enabling information retrieval based on the semantics of elements (things, not strings [3]). Consequently, the vision of accumulated human knowledge is nowadays pursued by the joint efforts of knowledge engineers, scientists, technicians, and crowdsourced contributions. While Semantic Web technologies focus on machine-interoperability of information and its processing, they are not aiming towards presenting the underlying information to humans [4], e.g., in the form of visualizations. However, visualizations support the human's ability to understand complex data through visual representations; *a picture is worth a thousand words*.

This thesis studies the visual exploration of Semantic-Web-Based knowledge structures. As the visual exploration, we denote different visualization methods and interaction techniques, addressing user-centered design for applications that facilitate understanding of Semantic-Web-Based knowledge structures. We characterize Semantic-Web-Based knowledge structures as structured and semantically enriched representations of knowledge using Semantic Web technologies, enabling intelligent machine agents to understand the *meaning* of information and infer new knowledge.

1.1 Motivation

The advances in digitization produce vast collections of data. Analyzing such collections without machine support would require an enormous amount of human labor and produce time-consuming endeavors. The developments in hardware and software engineering realize powerful systems for addressing such information organization and analysis tasks.

However, to benefit most efficiently, the data has to be machine-actionable, i.e., systems can access different datasets, perform information retrieval tasks, interlink data with other datasets and also infer new knowledge. The FAIR principles [5] (Findability, Accessibility, Interoperability, and Reuse) address machine-actionability as using "*the capacity of computational systems to find, access, interoperate, and reuse data with none or minimal human intervention*" [6]. Thus, these principles provide guidelines for publishing data, enabling manual and automatic discovery and reuse of datasets.

Before data becomes machine-actionable, systems require to understand its *meaning*, i.e., its semantics. Semantic Web technologies provide machine-readable representations, for example, in the form of the Resource Description Framework (RDF) [7], RDF Schema (RDFS) [8], or the Web Ontology Language (OWL) [9]. These representations encode *meaning* for information by realizing machine-readable descriptions with the capability for intelligent machine agents to access and interlink datasets and infer new knowledge using the semantics of the elements. Furthermore, various query languages, such as SPARQL [10], Gremlin [11], GraphQL [12], or Cypher [13] enable information retrieval, manipulation, and aggregation operations.

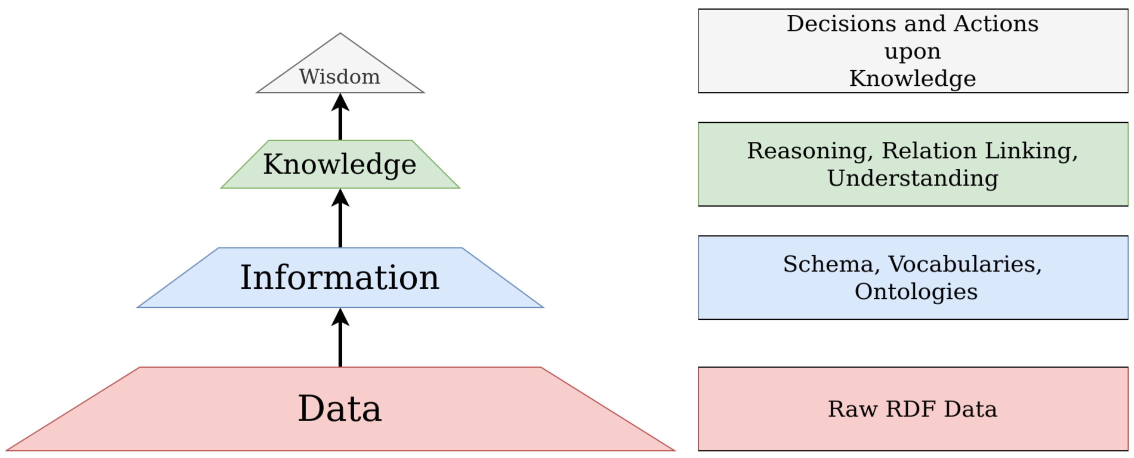

Wisdom
Knowledge
Information
Data
Decisions and Actions upon Knowledge
Reasoning, Relation Linking, Understanding
Schema, Vocabularies, Ontologies
Raw RDF Data

Figure 1.1: The DIKW pyramid aligning the individual aspects within the Semantic Web.

Different groups target the transformation of mere data towards information and knowledge by providing structured and semantically enriched formats that are understood by machines and humans alike. For example, DBpedia extracts structured information from Wikipedia pages and provides a uniform dataset that can be queried using Semantic Web technologies. Nevertheless, these transformations typically focus on machine-actionability, overlooking humans' involvement in designing such semantically rich structures.

Before we continue with the formalization of challenges in the visual exploration of Semantic-Web-Based knowledge structures, let us consider: what are data, information, and knowledge in Semantic Web contexts. Figure 1.1 shows the data, information, knowledge, and wisdom (DIKW) pyramid [14]. The data builds the foundation of the pyramid, whereas information extends data with schema, vocabularies, and ontologies. Knowledge is inferred using reasoning and interlinking of information. Finally, wisdom is obtained by acting upon knowledge. Thus, the Semantic Web represents data in machine-readable formats that serve as foundation for information organization and knowledge extraction.

A fundamental aspect of the Semantic Web is creating and communicating conceptualizations of information and data in various domains. Taxonomies, vocabularies, and ontologies serve this purpose by providing a formal and machine-readable representation of domain knowledge. The larger and more interlinked such vocabularies and ontologies become, the more challenging it is for humans to inspect and validate them. The main characteristic of Semantic Web data is its textual and structured representation format of information. While machines can quickly access and process such large amounts of data with varying formats, humans' cognitive capabilities can quickly reach their limitations due to information overload.

Various ontology modeling tools have been created to help knowledge engineers with developing ontologies (e.g., Protégé[1]) by providing all OWL formalization features and supporting their composition. However, these tools are challenging to learn for users new to ontology modeling. Additionally, the modeling of ontologies is frequently done collaboratively in joint efforts of knowledge engineers and domain experts. On the one hand, domain experts, providing the conceptualization of domain knowledge, are typically not familiar with semantic formalism and conceptual modeling techniques. They often find it hard to follow logical notations in OWL representations. On the other hand, knowledge engineers, providing the necessary know-how for ontology modeling and logical notations in OWL, usually lack the domain's expertise to create ontologies of sufficient quality [15]. As Semantic Web receives growing attention in industrial and academic contexts, user groups developing ontologies become more diverse, including knowledge engineers, domain experts, and other user groups with various backgrounds.

Visualizations provide an abstraction of information that reinforces human cognition and facilitate understanding of complex data through its visual representations. Thus, visualizations offer a good starting point for exploration with an additional cognitive support for the understanding of provided information [16]. In the context of Semantic-Web-Based knowledge structures, various visualization methods and tools are available, and new ones are being developed every year [17]. The applied methods range from indented trees and chord diagrams to treemaps and Euler diagrams. Suitable visualizations, however, are highly dependent on individual use cases and targeted user groups.

The challenge with most approaches is grounded in their design. On the one hand, visualization methods are created with a particular definition for the representation model. On the other hand, users perceive the provided visualization and build a mental model for the interpretation of the content [18]. Ideally, the visual model corresponds to the user's mental model. However, these match typically *only* in some aspects and diverge from user's expectations and previous experiences with other visualizations.

Figure 1.2 illustrates an overview of Semantic-Web-Based knowledge structures, their transformation to visualizations, and different visual representation methods. The data structures serve as the foundation for creating visualizations. Visualization generation addresses the process of creating visual representations for different use cases and user groups with their varying requirements. Visualizations serve as the means to exploit humans' ability to understand complex data through its visual representations.

[1]https://protege.stanford.edu/

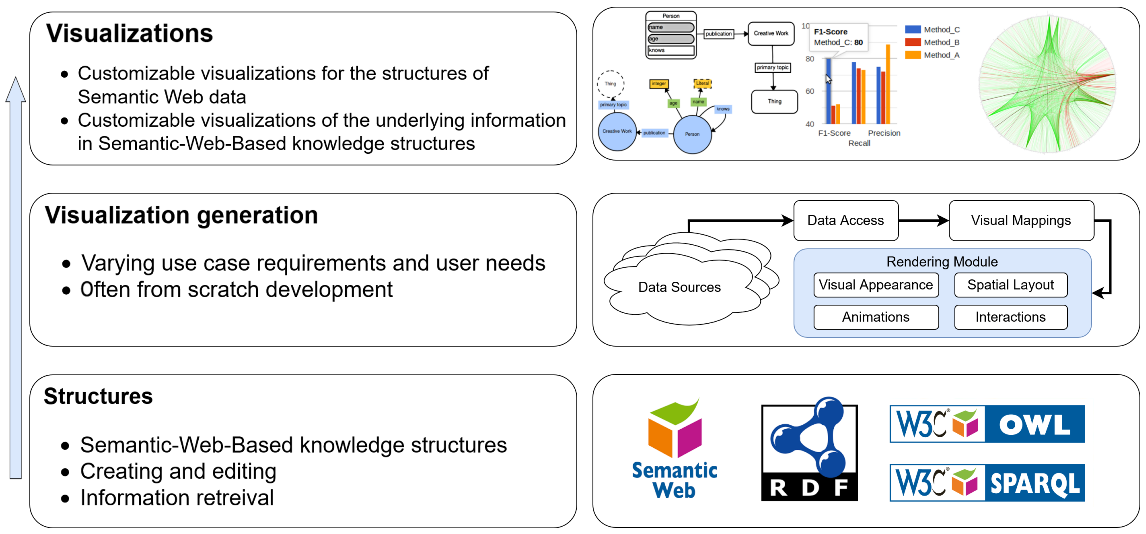

Visualizations
Customizable visualizations for the structures of Semantic Web data
Customizable visualizations of the underlying information in Semantic-Web-Based knowledge structures
Visualization generation
Varying use case requirements and user needs
Often from scratch development
Structures
Semantic-Web-Based knowledge structures
Creating and editing
Information retreival
Data Sources
Data Access
Visual Mappings
Rendering Module
Visual Appearance
Spatial Layout
Animations
Interactions
F1-Score
Precision
Recall
Method_C
Method_B
Method_A
Semantic Web
RDF
W3C OWL
W3C SPARQL

Figure 1.2: **Overview:** The lowest layer address different Semantic-Web-Based knowledge structures. The next layer indicates the challenges of creating visualizations from such structures. The top layer shows the different visual representation methods for different use cases. Semantic Web logos from [19].

This thesis studies how to facilitate understanding of Semantic-Web-Based knowledge structures through visual representations. In this thesis, we denote Semantic-Web-Based knowledge structures as structured and semantically enriched representations of knowledge using Semantic Web technologies, enabling access to the data using ontology files, SPARQL query results, and RESTful-API responses of Knowledge Graphs. We argue that in order to facilitate understanding of Semantic-Web-Based knowledge structures, customizable visual representations play a vital role in the successful realization of the visual exploration and the engagement of diverse user groups in different use cases.

1.2 Challenges and Problem Statement

While Semantic Web data typically reflects graph structures, visualizing such structures as node-link diagrams can quickly result in large and complex visual representations. More suitable visualization methods are required to address the data at hand and enable an effortless understanding of the underlying information. Visualizations have to be designed in a user-centered way to facilitate understanding and interaction with such data. In the following, we define challenges for the visual exploration of Semantic-Web-Based knowledge structures as three sub-sequential challenges, addressing the data structures, visual representations, and the generation process of visualizations for different use cases.

Challenge 1: Accessing and manipulating data from a user perspective

Versatile technologies are used to store and process data. However, this also implies different serialization and information retrieval formats. The data has various representations (even though using Semantic-Web-Based data structures such as RDF). The interfaces connecting to such data sources use multiple technologies and languages (e.g., SPARQL, RESTful-API, etc.). Therefore, the challenge lies in the interoperability of different data sources, the information retrieval results, and their capabilities to manipulate data, i.e., create new entries or modify existing ones. To facilitate accessing and editing of Semantic Web data, users require approaches that reduce the complexity of interactions with such data and lower the entry barriers for user groups less familiar with the technical details.

Challenge 2: Mapping data to visual primitives and its presentation

Having established a connection to a data source and retrieved some of its underlying information presents us with an additional challenge: The retrieved data has different serialization formats (e.g., CSV or JSON). Furthermore, it is not necessarily optimized for visual mappings. Thus, various data sources and information retrieval results require parsing mechanisms to create data models that are used as foundations for visual mappings. These visual mappings need additional model transformations to create various visualization methods, e.g., node-link diagrams, indented trees, and chart visualizations.

The presentation of information in the form of visualizations facilitates exploration and sense-making. However, different user groups have various backgrounds and expectations for a visual representation. The amount of visualization methods and tools in Semantic Web contexts emphasizes the need for representing the underlying graph structures in various ways for different use cases and user groups. Additionally, with the growing size and complexity of the data we wish to visualize, its representation can become hard to read and comprehend. Every rendering primitive (e.g., circle, rectangle, text, link, arrow, and even color) represents information. An information overload is a natural consequence when the amount of rendering primitives exceeds the cognitive and perceptive capacity of the user. Additionally, visualizing *large* graphs in the form of node-link diagrams results in decreased readability through visual clutter, crossing edges, and occlusion. Therefore, visualizations have to handle humans' limited cognitive capacity and mediate the information load for a simpler understanding of the underlying data.

Challenge 3: Varying requirements for visualizations

The varying requirements of visualizations do depend on the individual use cases and the targeted user groups. Thus, visualization methods and tools typically result in tailored-suited applications that serve the current needs of a particular scenario. However, reusing these created solutions becomes challenging when the targeted user groups and the use cases with their underlying requirements change. Furthermore, "*new visualization methods and tools are often developed from scratch, omitting opportunities to learn from previous mistakes or to reuse advanced techniques provided by other researchers and developers*" [17, pp. 1-2].

1.3 Research Questions

The main research question of this thesis addresses how to facilitate the visual exploration of Semantic-Web-Based knowledge structures. The previously presented challenges motivate the three sub-sequential research questions that are in the focus of this thesis:

Research Question 1 (RQ1)

How can we ease the creation and editing process of Semantic-Web-Based knowledge structures from a user perspective?

Creating Semantic-Web-Based knowledge structures typically involves joint efforts of knowledge engineers and domain experts. To engage different user groups and answer this question, we investigate how to decrease the entry barriers for involvement in ontology modeling from a user perspective. In particular, we define requirements for a device-independent visual modeling approach and demonstrate its benefits using two user studies.

Research Question 2 (RQ2)

How can we improve understanding of Semantic-Web-Based knowledge structures using interactive and user-centered visualizations?

Visualization methods are typically created with a particular definition for the representation model. However, users often require customizations to reflect their previous experiences with other visualization methods and tools. This question addresses how to create customizable visualization for Semantic-Web-Based knowledge structures. We present a methodology for customizable ontology visualizations, defining visual representation as meta ontologies, facilitating their exchange and reuse. Furthermore, we present customizable chart visualizations for tabular data originating from Knowledge Graphs.

Research Question 3 (RQ3)

How can we ease the creation of visual representations in Semantic Web contexts for different use cases and diverse audiences?

Reusing existing solutions usually requires additional development effort to address the use case and targeted user groups' needs. This question addresses how to facilitate the creation process of visual representations by building an open-source ecosystem that allows for reusing existing components, contributing to the code base, and facilitating adoption to the requirements at hand. We present a customizable pipeline-based approach for creating visual representations in Semantic Web contexts. Our approach applies the separation of concerns paradigm for the commonly used steps in the visualization generation process. We address a unified visualization framework through divergence in components and convergence in data models.

1.4 Thesis Overview

This section presents a high-level descriptive overview of achieved outcomes for the conducted research. We highlight the main contributions of the thesis and list references to published scientific articles completed during the research.

1.4.1 Contributions

Contributions for RQ1

Device-independent visual modeling in Semantic Web contexts.

Due to the increased attention of Semantic Web technologies in academic and industrial contexts, different user groups are involved in the creation of Semantic-Web-Based knowledge structures. Visual modeling approaches allow for engaging domain experts who are less familiar with semantic formalism and conceptual modeling techniques. Additionally, knowledge workers often use more than one device for their daily tasks in a multitude of interaction contexts, ranging from classical desktop settings to mobile scenarios in meetings, workshops, and on business trips. We identified devices, interaction methods, and requirements for a device-independent visual ontology modeling approach. Furthermore, we present two studies indicating the benefits of the approach for various user groups.

Contributions for RQ2

- GizMO – Graph Visualization Meta Ontology, enabling the customizable visual representations of ontologies using annotation ontologies.
- An approach for customizable chart visualizations of tabular data originating from Knowledge Graphs.

We present a methodology for creating definitions for visual representations using meta ontologies. We demonstrate its applicability by realizing a Graph Visualization Meta Ontology (GizMO) for node-link diagram visualizations. Additionally, GizMO is accompanied by two prototype implementations facilitating the creation of annotation ontologies. The conceptualization of containers (i.e., the domain ontology and meta ontologies) allows users to exchange visualizations directly within the framework. Furthermore, users can create new visual notations, reuse existing ones, and share them with others.

Tables are frequently used in academic and industrial contexts to provide an organized and compressed depiction of information. Thus, we present an approach for customizable chart visualizations of tabular data originating from Knowledge Graphs. Our approach augments Knowledge Graph representations of tabular data with additional semantics, enabling information organization and data transformations for selecting suitable visualizations.

Contributions for RQ3

- Refinement of commonly used steps in visualization generation using the separation of concern paradigm.
- Pipeline-based visualization framework addressing various data sources in Semantic Web contexts.
- Visual pipeline builder to facilitate initialization and configuration of source code infrastructures.

Visualizations are often tailor-suited to fulfill the underlying requirements of use cases. However, these solutions often become rigid and require modification with changing requirements and user needs. We provide a conceptualization for a unified visualization framework through divergence in components and convergence in data models. Our approach refines the visualization process using the separation of concern paradigm and a modular architecture. The open-source components and data models allow for reuse and further adjustments to the requirements at hand. A visual pipeline builder serves as an entry point to enable the swift creation of pre-configured source code infrastructures.

1.4.2 Publications

Different aspects of this thesis have already been published at conferences and workshops. The following references serve as a foundation for presenting approaches, ideas, figures, and tables for the later chapters. The list of all publications related or not related to this theses is presented in Appendix A.

1. **Vitalis Wiens**, Steffen Lohmann, Sören Auer. *WebVOWL Editor: Device-Independent Visual Ontology Modeling*. International Semantic Web Conference 2018 (P&D); **Best Demo Award**
2. Muhammad Rohan Ali Asmat, **Vitalis Wiens**, Steffen Lohmann. *A Comparative User Evaluation on Visual Ontology Modeling Using Node-Link Diagrams*. Emerging Topics in Semantic Technologies - ISWC 2018 Satellite Events, 1–12, IOS Press.
3. **Vitalis Wiens**, Mikhail Galkin, Steffen Lohmann, Sören Auer. *Demonstration of a Customizable Representation Model for Graph-Based Visualizations of Ontologies–GizMO*. International Semantic Web Conference 2019 (Poster & Demos); **Best Demo Award**.
4. **Vitalis Wiens**, Steffen Lohmann, Sören Auer. *GizMO–A Customizable Representation Model for Graph-Based Visualizations of Ontologies*. Proceedings of the 10th International Conference on Knowledge Capture 2019, 163–170, ACM.
5. **Vitalis Wiens**, Steffen Lohmann. *Demonstration of a Customizable Knowledge Graph Visualization Framework*. ISWC 2020 (P&D).
6. **Vitalis Wiens**, Markus Stocker, Sören Auer. *Towards Customizable Chart Visualizations of Tabular Data Using Knowledge Graphs*. The 22nd International Conference on Asia-Pacific Digital Libraries (ICADL 2020).

1.5 Thesis Structure

This thesis is divided into eight chapters which are outlined as follows:

- Chapter 1 covers the main challenges, research questions, the description of achieved contributions, and results with a list of published articles.

- Chapter 2 introduces the conceptualizations of Semantic-Web-Based knowledge structures, fundamental graph algorithms, and human factors in visualizations.

- Chapter 3 provides an overview of state-of-the-art approaches related to Semantic Web technologies and visualizations related to the main research question.

- Chapter 4 introduces requirements for a device-independent visual modeling approach, reducing entry barriers for engagement in modeling Semantic-Web-Based knowledge structures. Furthermore, we present two studies indicating the benefits of the approach for various user groups.

- Chapter 5 presents a methodology for customizable visualizations of ontologies that focuses on the utilization of OWL to define meta ontologies for visual representations and facilitate the creation, sharing, and reuse of existing visual notations.

- Chapter 6 presents an approach for customizable chart visualizations for tabular data originating from Knowledge Graphs. Additional semantics enable information organizations and visualization suggestions.

- Chapter 7 describes an approach that refines the commonly used steps in the visualization generation process. A pipeline-based architecture facilitates the creation of *the right components for the right task*. Our approach addresses a unified visualization framework through divergence in components and convergence in data models.

- Chapter 8 provides conclusion remarks of this thesis with an additional outlook on future work. The research questions are revised based on the obtained results.

CHAPTER 2

Background

This chapter introduces the foundations and conceptualizations that serve as reference points to the research objective addressed in this thesis. Section 2.1 presents the foundation of Semantic Web technologies. Section 2.2 provides an overview of conceptualizations related to graph theory for the underlying graph structures prevalent in Semantic-Web-Based knowledge representations. Section 2.3 discusses general aspects of visual representations.

2.1 Semantic Web Technologies

The *Web* presents an information management system that uses interlinked Hypertext Markup Language (HTML) documents. Clients (e.g., web browsers) send requests using the Hypertext Transfer Protocol (HTTP) to servers that host and provide access to those documents. While this information management system provides some capabilities for machine-readability, i.e., the structure of HTML documents, it does not enable intelligent machine agents to understand the content and the meaning of the underlying information.

The Semantic Web is an extension of the Web [20]. It provides a well-defined *meaning* and machine-readable representations for information. Therefore, the conceptualization of Semantic-Web-Based knowledge structures implements machine-readable and machine-actionable information representations that reflect network-like structures. Semantic Web data describes information in the form of resources and their interrelations, reflecting graph-based data structures. Additionally, the explicitly defined *meaning* for all elements enables automatic reasoning and information processing.

As shown in Figure 2.1, the Semantic Web uses a technology stack that is organized in multiple layers. The lowest layer presents the conceptualization for identifying resources using Unique Resource Identifiers (URIs) and related character encoding. The Internationalized Resource Identifiers (IRIs) complement URIs by realizing the Universal Character Set (Unicode/ISO10646) for identifying resources. For the sake of simplicity, throughout this thesis, we will use URI as the term denoting the identification of resources. The next layer in the stack is the Extensible Markup Language (XML) [21]. It provides a general syntax for describing structured information. Other syntaxes, such as Turtle, N3, RDFa, and JSON-LD, provide machine-readable formats for different use cases.

The Resource Description Framework (RDF) presents the technology stack's core layer, which we describe in Section 2.1.1 in more detail. RDF provides a formal definition of resources using statements in the form of a triple pattern <*subject predicate object*>. Initially designed to process meta information, RDF has been generalized to provide the formal semantics for representing information on the Web using a standardized vocabulary [22].

The following layers in the technology stack address the extension of RDF. RDF Schema (RDFS) provides semantics for hierarchical structures. The Web Ontology Language (OWL) presents additional means to create formalizations of complex ontologies for information processing [4]. Semantic Web Rules Language (SWRL) defines the formal semantics for logical rules, such as causal relations. The standardized *SPARQL Protocol and RDF Query Language* allows for creating semantic queries retrieving and manipulating information stored in RDF data models and Knowledge Graphs.

The following two layers of the stack, i.e., Unifying Logic and Proof, address the realization of machine reasoning and intelligent machine agents using formal semantics based on Description Logic, which is based on First-Order Logic. However, no standardization for these two layers exists yet.

Similarly, the top layers consisting of trust, cryptography, and user interface also do not have standardizations. This thesis focuses on the user interface and application layer to realize visualizations facilitating interaction and understanding of Semantic-Web-Based knowledge structures.

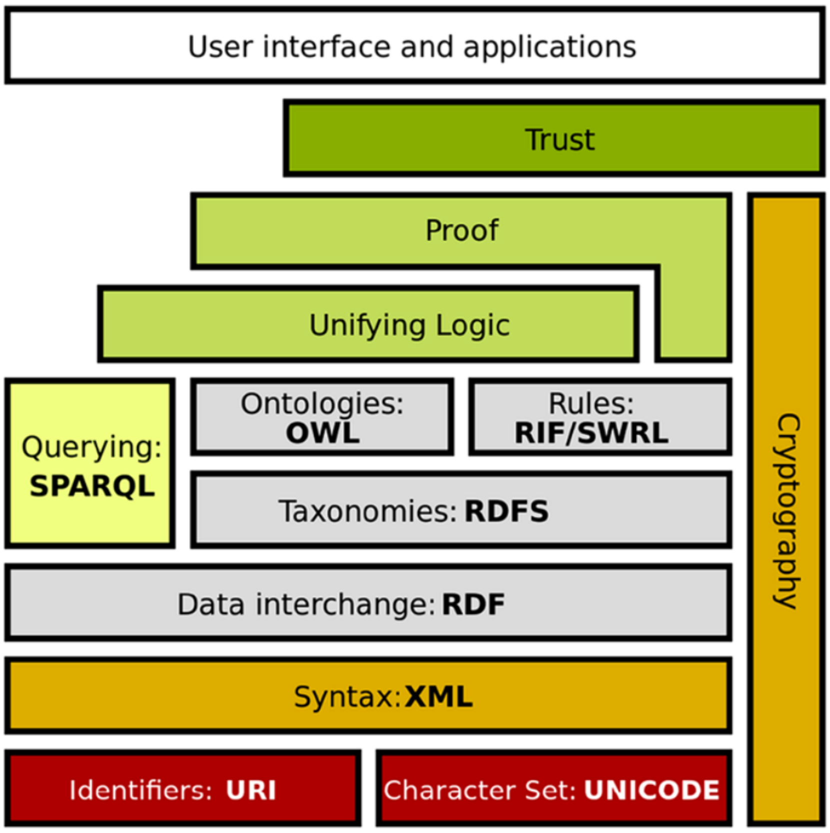


Figure 2.1: The Semantic Web technology stack [23].

2.1.1 Resource Description Framework

The Resource Description Framework (RDF) is a standard of the World Wide Web Consortium (W3C[1]). RDF defines an abstract syntax and serialization formats such as Turtle and JSON-LD for storing and exchanging RDF-based data. W3C provides additionally the standardized RDF Schema (RDFS) vocabulary, which is an extension of the basic RDF vocabulary for data modeling. Furthermore, W3C provides the SPARQL Query Language standard for manipulating and retrieving RDF datasets.

RDF uses a graph-based data model for the semantically enriched representation of information on the Web. A set of triples in the form of <*subject predicate object*> defines an RDF graph. The *subject* and the *object* reflect nodes in a graph. The *predicate* connects these nodes through a directed link. Figure 2.2 shows a node-link diagram representation for an example statement: *The Millennium Falcon is a spaceship*. The subject of the triple `ex:Millennium_Falcon` is a resource URI. The predicate `rdf:type` is a URI

[1]https://www.w3.org/standards/semanticweb/

defining a semantic relation *"is a"*. The object `ex:Spaceship` is a resource URI. The prefixes `ex:` and `rdf:` implement abbreviations of these URIs. The shortened version of the URI `ex:Millennium_Falcon` translates to its full version by expanding the prefix, i.e., `http://example.org/Millennium_Falcon`.

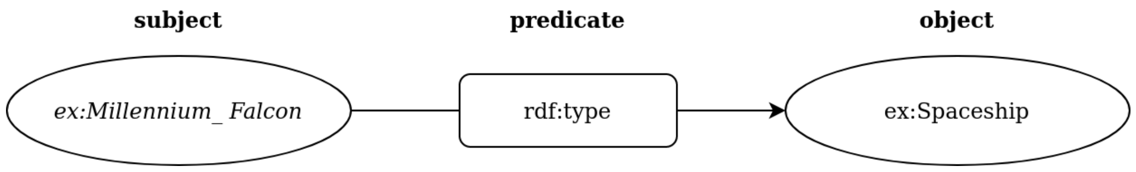


Figure 2.2: An illustration of an example RDF triple as a node-link diagram: The predicate reflects a directed link between the subject and the object node.

The RDF graph data model supports three types of nodes, i.e., URIs, literals, and blank nodes. The resource denoted by a URI is also called its referent [7]. URI is an extension of Unique Resource Locator (URL), which often indicates locations of documents on the Web. For example, the URI `http://example.org/Millennium_Falcon` provides us with an identifier but does not reflect a document on the Web. Thus, not every URI denotes a Web document. URIs typically have the following hierarchical structure [24]:

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
```

Literals represent values such as strings, numbers, and dates. Literals can contain multiple elements, e.g., the lexical form and optional language tags or datatype URIs. Typed literals, such as numbers and dates, refer to XML Schema [25] datatypes providing operational semantics. For example, the typed literal `"4.2"^^xsd:float` extends the lexical form of 4.2 with an additional float datatype. Similarly, language tags according to BCP 47 [26] extend strings with semantics for corresponding languages, e.g., `"Световой меч"@ru` and `"Lichtschwert"@de` are translations of the word "Lightsaber" in Russian and German, respectively. If present, the language tag or a datatype URI provides explicit meaning for the lexical form. Otherwise, the lexical form is interpreted as a plain string. Literals can *only* occur at the position of the *object* in a triple.

Blank nodes define resources without identifiers and are disjoint from URIs and literals. While blank nodes do not have identifiers in the abstract syntax of RDF, using a concrete syntax (e.g., Turtle, RDFa, etc.) introduces identifiers with *only* a local scope that are merely artifacts of the serialization [27]. Blank nodes are often used to define enumerable data structures such as lists, containers, and collections. For example, describing faculty members can be modeled as a blank node with links to individual members (cf. Figure 2.3).

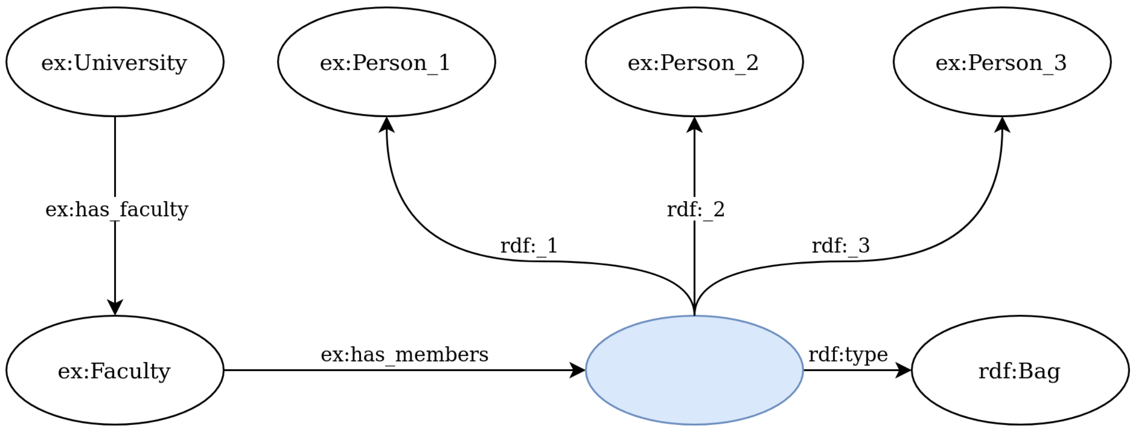


Figure 2.3: The realization of a container for faculty members using a blank node (blue).

Figure 2.3 illustrates an example for modeling an unordered container using a blank node. The type assertion for the blank node is `rdf:Bag`, a class of a container indicating that its members are unordered [8]. The properties `rdf:_1`, `rdf:_2`, and `rdf:_3` state the membership of the resources belonging to the container. Sorted containers are realized through the type assertion of the blank node to the class `rdf:Seq`. The type assertion `rdf:Alt` (alternative) indicates that typically one of the members will be selected [8].

Collections reflect a list of items. Similar to implementations in various programming languages, an item is defined as a blank node that points to a resource using the property `rdf:first`. The property `rdf:rest` points to the next item in the collection. The resource `rdf:nil` (not in list) is an instance of type `rdf:List` and reflects an empty list. Figure 2.4 shows the example for faculty members modeled as a collection.

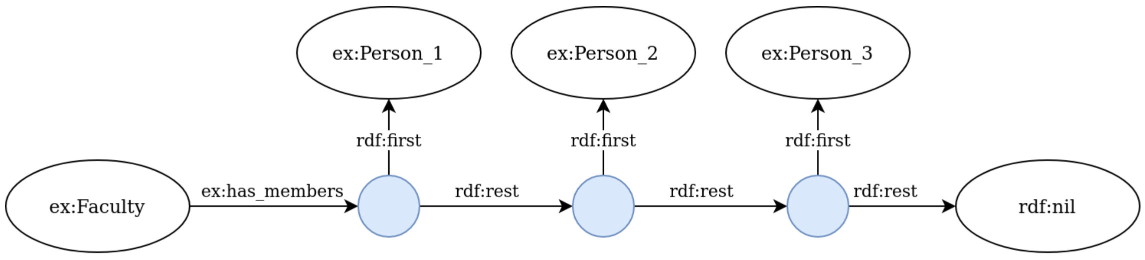


Figure 2.4: Modeling of the faculty members as a collection. Blank nodes are highlighted in blue.

The distinction between *collections* and *containers* is grounded in different use cases. While containers provide no mechanisms to identify a certain amount of members, collections can define a closed set of members [8]. Thus, containers reflect ordered and unordered composition of items with the additional option to add new elements. Collections reflect ordered compositions of items and restrict the possibility of adding new elements.

The main building block of RDF are triples (i.e., <*subject predicate object*>), where:

- **Subject** denotes the element from which a directed connection is formed to the object using the predicate. Only URIs or blank nodes can be placed at the subject position.
- **Predicate** denotes the element that forms the directed connection between subject and object. Only URIs can be placed at the predicate position.
- **Object** denotes the target element for the connection. URIs, blank nodes, or literals can be placed at the object position.

Formally, an RDF triple is defined as follows:

Definition 2.1: RDF Triple [28]

Let $\mathbf{U}$, $\mathbf{B}$, $\mathbf{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. A tuple $(s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple, where s is called the subject, p the predicate, and o the object.

2.1.2 Extended Data Modeling with RDFS and OWL

The RDF vocabulary provides mechanisms to make statements about resources in the form of RDF triples. However, this vocabulary supports basic descriptions, e.g., *x has type y*, and does not provide the means to model groups of resources, such as classes. RDF Schema (RDFS) extends the basic RDF vocabulary with data-modeling aspects [8].

RDFS provides the means to describe groups of resources, their relationships, and hierarchical structures. Furthermore, RDFS defines characteristics of resources and properties, such as the domain and range restrictions. RDFS uses an object-oriented conceptualization for classes and properties as often used in programming languages such as Java. In contrast to object-oriented systems, which define a class in terms of properties its instance may have, RDFS defines properties in terms of classes for a resource to which they apply [8].

Hierarchical structures for classes are realized using the predicate `rdfs:subClassOf`. The predicate `rdfs:subPropertyOf` addresses the hierarchy of properties. Property resources define the additional restrictions, e.g., using `rdfs:domain` and `rdfs:range` properties. These encode additional semantics. While the domain assigns a class to the subject, the range assigns a class to the object. The RDFS vocabulary defines additional complementary predicates, such as `rdfs:label` and `rdfs:comment`, to provide natural text annotations for resources. The example in Listing 2.1, expressed using the Turtle (TTL) syntax, illustrates the use of RDFS for defining class and property hierarchies.

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:     <http://www.example.org/> .

ex:Spaceship    rdfs:subClassOf     ex:Vehicle .
ex:Pilot        rdfs:subClassOf     ex:Person .
ex:Passenger    rdfs:subClassOf     ex:Person .

ex:isPiloting   rdfs:domain         ex:Pilot .
ex:isPiloting   rdfs:range          ex:Vehicle .
ex:isPiloting   rdfs:subPropertyOf  ex:operates .
```

Listing 2.1: RDFS example defining subclass axioms for `Spaceship`, `Pilot`, `Passenger`, and defining domain, range, and hierarchical relation for the property `isPiloting`.

Using RDFS, we are able to express that every `Spaceship` is a `Vehicle`. Both `Pilot` and `Passenger` are `Persons`. The property `isPiloting` is a specialized action of `operates`. RDFS enables basic inference. For example, using the triple <`ex:Han_Solo ex:isPiloting ex:Millennium_Falcon`> and the vocabulary of Listing 2.1, a reasoner can infer by using the domain and range that `Han Solo` is a `Pilot`, whereas `Millennium Falcon` is a `Vehicle`. Without an explicit assignment of `Millennium Falcon` to `Spaceship` this information can not be entailed.

RDFS often accompanies the definition of more complex ontologies using the Web Ontology Language (OWL). The term *ontology* is borrowed from philosophy addressing the science of describing entities in the world and how they relate [29]. As defined by Gruber [30, p. 908], "*an ontology is an explicit specification of a conceptualization*". Furthermore, "*a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose*" [30, p. 908].

OWL extends RDF and RDFS vocabularies with additional logical constructs and formal semantics. In the following, we describe a subset of new constructs and their semantics:

- Class level semantics:
  - `owl:equivalentClass` states the equivalence of two or more classes.
  - `owl:unionOf` states that a class definition is a union of multiple classes.
  - `owl:intersectonOf` states that a class is an intersection of multiple classes.
  - `owl:disjointWith` states the disjointedness of a class to other classes.

- Instance level semantics:
  - `owl:sameAs` states the equivalence of two or more instances.
  - `owl:differentFrom` states difference of two or more instances.
- Predicate level semantics:
  - `owl:objectProperty` object value assertion for the object position.
  - `owl:datatypeProperty` literal value assertion for the object position.
  - `owl:cardinality`, `owl:allValuesFrom`, `owl:someValuesFrom` define cardinality restrictions, universal and existential quantifiers, respectively.

```
 1 @prefix owl:<http://www.w3.org/2002/07/owl#> .
 2 @prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 3 @prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
 4 @prefix ex: <http://www.example.org/> .
 5
 6 ex:Spaceship     owl:disjointWith ex:Person .
 7 ex:siblingOf     rdf:type         owl:ObjectProperty,
 8                                   owl:SymmetricProperty .
 9 ex:siblingOf     rdfs:domain      ex:Person .
10 ex:siblingOf     rdfs:range       ex:Person .
11 ex:Luke          ex:siblingOf     ex:Leia .
```

Listing 2.2: A small knowledge base. *Note:* Multiple type assertions are realized using a comma, see lines 7 and 8. A parser will unroll these abbreviations and create corresponding triples.

The example in Listing 2.2 defines a small knowledge base in the Turtle syntax. The OWL extensions enable logical formalizations and allow reasoners to deduce implicit knowledge. While it is evident that a person can not be simultaneously a spaceship, OWL explicitly encodes this information using the disjoint property. Given the triple <`ex:Luke ex:siblingOf ex:Leia`> and the RDFS semantics, we can deduce that `Luke` and `Leia` are of type `Person`. The domain and range restrictions on the property `ex:isSiblingOf` enable this entailment. However, this small knowledge base only defines the sibling relation from `Luke` to `Leia`. Generally, the sibling relation should describe: if X is a sibling of Y, then Y is a sibling of X. Using the semantics of OWL, we can encode this conceptualization by assigning an object property expression to the `ex:isSiblingOf` property in the form of `owl:SymmetricProperty` [9]. Thus, reasoners can entail the implicit knowledge that `Leia` is a sibling of `Luke`.

2.1.3 Linked Open Data and SPARQL

Accessing structured and interlinked data plays a vital role in the success of Semantic-Web-Based knowledge structures. A major conceptualization of the Semantic Web addresses the interlinking of data so that humans and machines are able to explore the web of data (*"The Semantic Web isn't just about putting data on the web"* [31]). Linked Data addresses the vision of the Semantic Web by a set of best practices that are used in real-world use cases. The Linked Data principles have been defined by Sir Tim Berners-Lee as follows:

Definition 2.2: Linked Data Principles [31]

- Use URIs for identifying and naming things.
- Use HTTP URIs to allow for the lookup of things.
- URI lookups should provide useful information using the standards (RDF*, SPARQL).
- Link to other URIs to enable humans and machines to explore more things.

These principles encourage people to provide open and interlinked datasets for humans and machines alike. While Linked Data does not necessarily need to be open, openness promotes the reuse and interlinking of data. Linked Open Data (LOD) denotes Linked Data which is released under an open license. These principles and open license publishing led to the emergence of the LOD Cloud, a network of open interlinked semantic datasets containing 1,301 datasets with 16,283 links (as of May 2021) [32].

The sheer volume of data makes it cumbersome when serving the data as lots of files [31]. Thus, SPARQL query services provide reasonable mechanisms for information retrieval tasks. Examples of large community-supported open knowledge bases are DBpedia [33] (a Semantic Web representation of Wikipedia) and Wikidata [34] (a free and open knowledge base that acts as central storage for the structured data of projects such as Wikipedia).

SPARQL is a W3C recommendation and is a recursive acronym for SPARQL Protocol and RDF Query Language [10]. It allows for creating semantic queries retrieving and manipulating information stored in RDF data models and Knowledge Graphs. A SPARQL query is typically a *basic graph pattern* that is a set of triple patterns, similar to RDF triples. However, in a triple pattern, each subject, predicate, and object can be a variable. The triple patterns are matched against the target RDF dataset resulting in a subgraph where the variables are replaced with RDF terms when they form a query solution. Formally, a triple pattern and a basic graph pattern are defined as follows:

Definition 2.3: Triple Pattern, Basic Graph Pattern [35]

Let $\mathbf{U}, \mathbf{B}, \mathbf{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively.
Let $\mathbf{V}$ be a set of variables such that $\mathbf{V} \cap (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L}) = \emptyset$.
A triple pattern $tp_i \in (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{L} \cup \mathbf{V})$.
A Basic Graph Pattern (BGP) B is the conjunction of triple patterns,
i.e., $B = tp_1$ AND tp_2 AND ... AND tp_n.

SPARQL defines four query types: `SELECT`, `ASK`, `CONSTRUCT`, and `DESCRIBE`.

- `SELECT` returns variables and their bindings that satisfy the basic graph pattern.
- `ASK` query returns a boolean value if the basic graph pattern has a solution.
- `CONSTRUCT` returns an RDF graph that is constructed based on the BGP.
- `DESCRIBE` returns an RDF graph containing RDF data about resources.

SPARQL provides functionalities such as aggregation, logical operations, filtering, and optional patterns to create more complex queries. Listing 2.3 shows a simple query example that returns all triples of the target RDF dataset. The `SELECT` clause defines variables for the result, and the `WHERE` clause defines the generic triple pattern, matching all triples.

```
SELECT ?subject ?predicate ?object WHERE {
    ?subject ?predicate ?object .
}
```

Listing 2.3: A simple SPARQL SELECT query to retrieve all triples of the target RDF dataset.

Semantic Web data is typically realized in a versatile technology stack addressing different use cases. Knowledge Graphs use various technologies to store and retrieve information such as SPARQL endpoints and RESTful API requests with different query languages such as SPARQL, Cypher, and GraphQL. When designing visual representations, the returned types of the different query formats need to be considered. The main characteristic of the results is that these are transmitted in textual formats such as JSON, XML, or textual representations of an RDF graph. Thus, parsing mechanisms are required for transforming the results into visual primitives.

2.1.4 Knowledge Graphs

The term "*Knowledge Graph*" has existed in the literature since 1972 [36]. The modern association of this term originates from the announcement of the Google Knowledge

Graph [3]. Since then, the term is often used in relation to graph-based collections of knowledge. Prominent examples for public Knowledge Graphs are DBpedia [33], Wikidata [34], and YAGO [37]. Other companies such as Netflix, Amazon, Microsoft, and Facebook have created their own Knowledge Graphs similar to the one of Google.

The definition of the term "*Knowledge Graph*" remains ambiguous. The work of Ehrlinger and Wöss [38] argues that several definitions of Knowledge Graphs and their features exist. However, a unified definition of Knowledge Graphs is yet missing. Typically, Knowledge Graphs refer to large collections of semantically enriched, interlinked, and structured information. Instead of defining a formal definition of Knowledge Graphs, the work of Paulheim [39] presents a minimum set of characteristics:

Definition 2.4: Knowledge Graph [39]

- mainly describes real-world entities and their interrelations, organized in a graph
- defines a possible schema for classes and relations of entities
- allows for the interrelation of arbitrary entities with each other
- covers various domains

These characteristics address the structure and the content of Knowledge Graphs, i.e., facts about real-world entities structured in the form of a graph. The possible definitions of a schema for classes and relations, e.g., in the form of ontologies or vocabularies, provide the means to retrieve entities based on their semantics and also infer knowledge that is not explicitly stated in the Knowledge Graph. Additionally, as Knowledge Graphs do not pose any restriction on the information they hold, various knowledge domains are covered. Examples of different knowledge domains are Biomedical Knowledge Graphs [40], Enterprise Knowledge Graphs [41], Scientific Knowledge Graphs [42], and many more.

Figure 2.5 shows a real-world example of entities and their interrelations. This figure illustrates a small selection of facts extracted from DBpedia about Barack and Michelle Obama in the form of a node-link diagram. While we keep the amount of information selective for illustrative purposes, i.e., we depict *only* 11 triples, this example highlights once more that with the growing size of the data we wish to visualize, an information overload quickly materializes.

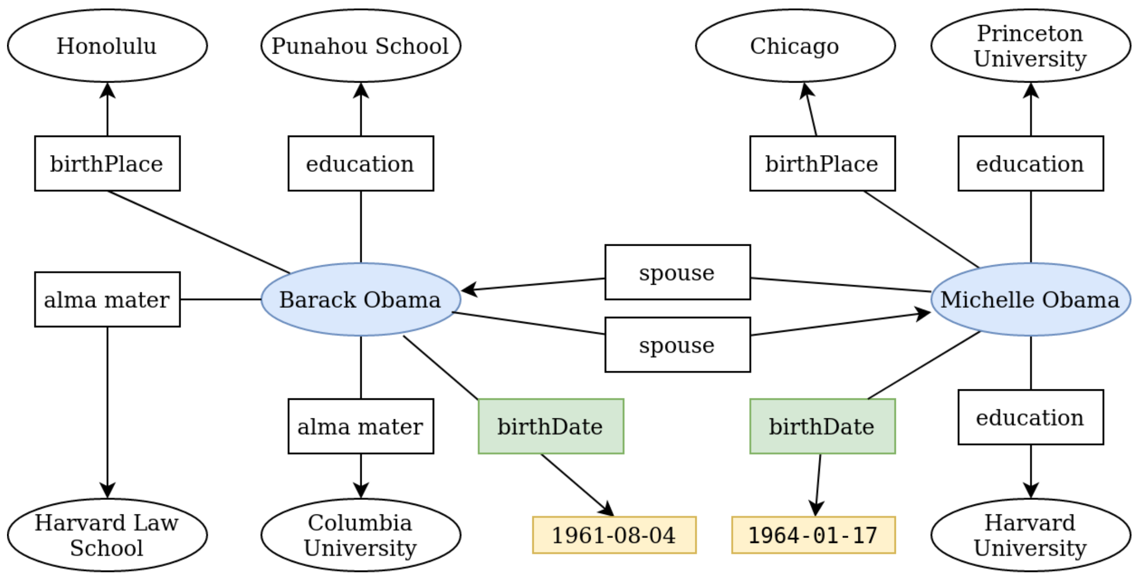


Figure 2.5: A node-link diagram visualization depicting a small selection of facts extracted from DBpedia about Barack and Michelle Obama. While the blue color highlights both persons, the green and yellow colors indicate the datatype properties and corresponding literal values.

The characteristics of Definition 2.4, however, also apply to the definition of ontologies. The conceptualizations of ontologies can be categorized into two groups of statements, i.e., *TBox* and *ABox*. The TBox describes the terminological aspects of the conceptualization, i.e., the definition of a schema, classes, properties, and their hierarchical structures. The ABox describes the assertion aspects of the conceptualization, i.e., the definition of instances/individuals assigned to classes and their interrelations. Thus, the first two characteristics are also addressed by ontologies. Similarly, the latter two characteristics are covered by interrelations of entities, e.g., definitions of properties connecting entities within an ontology or connecting entities outside the ontology using import statements for conceptualizations of different domain knowledge. These characteristics define Knowledge Graphs by their content, i.e., mainly describing real-world entities using graph-based knowledge representations. Other definitions for Knowledge Graphs address further technological capabilities of such systems and the underlying size of stored knowledge [38]. For example, information retrieval capability (e.g., using SPARQL) and inference and derivation of new interrelations using reasoning.

As stated before, Knowledge Graphs contain large collections of semantically enriched data stored in the form of a graph. In order to get a feeling of what large in this context means, we list certain statistics about popular Knowledge Graphs. DBpedia's dataset "2016-10" contains 13 billion pieces of information (RDF triples) [43]. Wikidata contains more than 90 million items [44] and 12 billion RDF triples [45]. YAGO (Yet Another Great Ontology) is another example large Knowledge Graph which contains 2 billion triples about 64 Million entities [46]. When Google's Knowledge Graph was announced in 2012, it contained 500 million objects and more than 3.5 billion relationships between them [3]. A news article [47] reported that Google's Knowledge Graph tripped its size within seven months after it was announced, containing 570 million entities and 18 billion facts.

As human cognitive capacities are limited, we require mechanisms to extract pieces of information and exploration techniques to navigate in these extensive collections of information stored in Knowledge Graphs. In the following example, we address a natural language question that we answer using a Knowledge Graph: "Who was born on the same day as Nikola Tesla?". To answer this question, we use DBpedia as source Knowledge Graph. As shown in Listing 2.4, a SPARQL query retrieves the answer, i.e., *Otto Krause*.

```
1 PREFIX dbo: <http://dbpedia.org/ontology/>
2 PREFIX dbr: <http://dbpedia.org/resource/>
3 PREFIX schema: <http://schema.org/>
4
5 SELECT DISTINCT ?person ?birthDate WHERE {
6    dbr:Nikola_Tesla dbo:birthDate ?birthDate .
7    ?person a schema:Person .
8    ?person dbo:birthDate ?birthDate
9 }
```

Listing 2.4: A SPARQL query to retrieve the persons who were born on the same day as Nikola Tesla.

This example illustrates an application over Knowledge Graphs, i.e., Question Answering. Question Answering (QA) addresses information retrieval aspects combined with Natural Language Processing (NLP) [48]. In order to obtain our results, we need to transform our natural language question into a query (here SPARQL). While the NLP aspect of QA automatically performs the transformations of questions into queries, in the following, we provide the manual transformation of our inquiry into a SPARQL query. At first, we extract the birth date of Nikola Tesla, i.e., line 6 in Listing 2.4. Line 7 builds a basic graph pattern (BGP) that retrieves all resources of type `schema:Person` and assigns them to the variable `?person`. Next, the BGP is extended to match only resources that have a connection to the variable `?birthDate` using the property `dbo:birthDate`. Thus, using this SPARQL query allows us to retrieve all persons who are born on a particular date. The retrieved results are shown in Table 2.1.

Table 2.1: Corresponding results of the SPARQL query of Listing 2.4

person	**birthDate**
`http://dbpedia.org/resource/Nikola_Tesla`	1856-07-10
`http://dbpedia.org/resource/Otto_Krause`	1856-07-10

This thesis refers to the term Semantic-Web-Based knowledge structures as the general conceptualization of machine-readable representation of knowledge in the form of semantically enriched graph-based data models. The particular implementation, the size of the underlying knowledge, and the used technology stack play only a minor role in this context. The main aspect we consider for Semantic-Web-Based knowledge structures is that information is represented in graph-based data models that provide well-defined meanings and machine-readable access using Semantic Web conceptualizations. The second central aspect addresses the capabilities to retrieve portions of the information stored in a Knowledge Graph and the integration of information across different ontologies (e.g., using `owl:imports` statements). The objective of visual explorations in the context of this thesis is to facilitate the understanding of knowledge that is represented in the form of graph-based data models. Thus, the data structure and its semantics are the foundation for the generation of visual representations.

2.2 Graph Theory

The study of Semantic-Web-Based knowledge structures requires an analysis of the underlying data from the graph theory aspect, i.e., the graph-based data model used to represent knowledge in a machine-readable manner. This section provides a brief overview of selected definitions in this context. Graphs are used to model pairwise relations between objects. Thus, graphs consist of nodes and links reflecting objects and their connections. Other notations use *vertices* or *points* for nodes; and *edges* or *lines* for links. In this section, nodes and links are the terms denoting objects and their relations, respectively.

A typical graph $G(N, L)$ is a tuple, where N and L are finite sets. The elements of N are the nodes of the graph, while set L reflects the links describing the interrelation of nodes. The properties of nodes and links define characteristics for a graph. The work of Korte and Vygen [49] provides an overview of definitions and characteristics related to graphs which we summarize and adjust to our notation for the sake of simplicity in the following:

Definition 2.5: Undirected Graph [49]

An *undirected graph* is a triple $G_{undirected} = (N, L, \Psi)$,
where N and L are finite sets and $\Psi : L \rightarrow \{X \subseteq N : |X| = 2\}$.

The mapping for the links Ψ is restricted to $|X| = 2$, declaring that there exists a link ($l \in L$) connecting *only* two nodes ($n_1, n_2 \in N$). The links in an undirected or bidirectional graph describe the relation between two nodes, whereas the order of the nodes is not relevant, i.e., $l = (n_1, n_2) = l' = (n_2, n_1)$.

Definition 2.6: Directed Graph [49]

A *directed graph* is a triple $G_{directed} = (N, L, \Psi)$,
where N and L are finite sets and $\Psi : L \rightarrow \{(n_1, n_2) \in N \times N : n_1 \neq n_2\}$.

In contrast to undirected graphs, a link $l = (n_1, n_2)$ defining the connection from n_1 to n_2, does not imply the connection from n_2 to n_1 in a directed graph.

Definition 2.7: Parallel Links [49]

Two links l, l' are called *parallel* when $\Psi(l) = \Psi(l')$.

Parallel links occur when two nodes $n_1, n_2 \in N$ are connected using different links. For example, in a directed graph, two links are parallel when their source and target nodes are identical. Lines 8 and 9 in Listing 2.2 indicate that the predicates `rdfs:domain` and `rdfs:range` are parallel in this example because the subjects and the objects of these two triples are identical, i.e., `ex:isSiblingOf` and `ex:Person`.

Definition 2.8: Simple Graph and Multigraph [49]

A graph is called *simple* when it does not contain parallel links;
otherwise, it is called a *multigraph*.

A link in a simple graph is usually identified with its image $\Psi(l)$ and the graph G is denoted as $G = (N(G), L(G))$, where $L(G) \subseteq \{X \subseteq N(G) : |X| = 2\}$.

Definition 2.9: Self-Loops [49]

Links connecting identical nodes ($l = (n_i, n_i)$) are called self-loops.

An example for self-loops in RDF contexts is <`ex:Person ex:knows ex:Person`>. The property `ex:knows` forms a relation between identical subject and object resources.

Definition 2.10: Adjacency Matrix [49]

Adjacency Matrix A is a $|N| \times |N|$ Matrix, where $|N|$ is the number of nodes in the graph. The entry $A(i, j)$ reflects the number of connections between the nodes n_i and n_j. The matrix is symmetric if the graph is bidirectional.

Nodes that are directly connected through a link are called adjacent or neighboring. Thus, a graph structure can be represented in the form of an adjacency matrix A. In a simple graph, an entry $A(i, j)$ is *zero* when two nodes are not adjacent, and *one* when a link connects the nodes n_i and n_j. In a multigraph, the entry $A(i, j)$ often reflects the number of links connecting the nodes n_i and n_j.

Definition 2.11: Orientation [49]

For a directed graph G, it is sometimes beneficial to consider the underlying undirected graph G', which has the same set of nodes and contains an undirected link $\{v, w\}$ for each link (v, w) in G. In this case, G is considered as an *orientation* of G'.

The benefit of bidirectional links allows for additional flexibility for graph traversal, enabling navigation and exploration mechanisms that are not restricted by links' directions.

Definition 2.12: Subgraph [49]

A subgraph of a graph $G = (N(G), L(G))$ is a graph $H = (N(H), L(H))$ with $N(H) \subseteq N(G)$ and $L(H) \subseteq L(G)$. Thus, the graph G contains the subgraph H.

A subgraph H holds a subset of nodes and a subset of links of the original graph G. Subgraphs provide the means for visual exploration to reduce the information load. Additional exploration mechanisms allow for dynamically visualizing subgraphs, enabling users to collapse and expand nodes of a subgraph.

Definition 2.13: Spanning Graph [49]

A subgraph H of G is called *spanning* when it contains all nodes of the graph G, i.e., $N(H) = N(G)$. Thus, a spanning graph H differs only in the set of links.

Spanning Graphs provide the means to mediate relational information between the nodes of a graph G. In a multigraph, these allow for reducing the amount of visualized relations between node pairs. Visual explorations mechanisms allow for enabling users to collapse and expand links.

Definition 2.14: Path [49]

A *path* is a graph $P = (\{n_1, ..., n_{k+1}\}, \{l_1, ..., l_k\})$ such that $n_i \neq n_j$ for $1 \leq i < j \leq k+1$ and the sequence $n_1, l_1, n_2, ..., n_k, l_k, n_{k+1}$ is called a walk.

Paths describe the graph traversal using a walk from n_1 to n_{k+1}. Using additional weights for the links allows for the formalization of traversal costs. These traversal costs can be beneficial for identifying paths with minimal traversal costs.

Definition 2.15: Connected and Unconnected Graphs [49]

Let G be an undirected graph. G is called *connected* if there is a v-w-path for all $v, w \in N(G)$. Thus for all pairs of two nodes v, w there exists a path where a walk exists from v to w; otherwise, the graph is called *disconnected*.

A directed graph is called connected if the underlying undirected graph is connected. The connected components of a graph G are its maximal connected subgraphs. A link connecting two connected components is also called a *bridge*.

Definition 2.16: Cyclic Graph [49]

A graph with a circuit is called a *cyclic* graph.

A circuit (cycle) is a graph $(\{n_1, ..., n_k\}), (\{l_1, ..., k_k\})$, where the sequence $n_1, l_1, n_2, ..., n_k, l_k, n_1$ and $n_i \neq n_j$ for $1 \leq i < j \leq k$, is a walk with an identical start and end node.

Definition 2.17: Labeled Graph [49]

A labeled graph G is a graph $G(N, E, \Sigma, \Lambda)$, where $\Sigma(N)$ maps the set of nodes to the set of labels, and $\Lambda(L)$ maps to the set of links to the corresponding set of labels.

Labeled graphs assign to the sets of nodes or edges corresponding sets of labels. Further specifications of labeled graphs are node-labeled and link-labeled graphs, addressing the existence of corresponding label sets.

The definitions and characteristics above allow us to describe the formal definition for the RDF graph data model. An RDF graph is a directed graph that is defined by the triple structure <*subject predicate object*>, where the predicate is a directed link describing the interrelation of the subject to the object. RDF allows for the definition of arbitrary triples linking subjects and objects through different predicates. This can result in parallel links, which is a characteristic of a multigraph. For example, lines 8 and 9 in Listing 2.2 link identical subjects and objects using two distinct properties, i.e., `rdfs:domain` and `rdfs:range`. Furthermore, the triple structure allows for the definition of circuits within a graph, which is a characteristic of a cyclic graph. An RDF graph is a labeled graph, where the sets of labels for nodes and links are created using the corresponding URIs. To increase readability for humans, the labels in a visualization can be replaced with natural text labels using the annotations of resources such as `rdfs:label`. However, such labels are ambiguous (i.e., distinct resources can have the same natural text labels). Thus, the underlying labels for the machine operations remain the URIs of resources. Blank nodes receive a local scope label through the realization, which is an artifact of the serialization (cf. Section 2.1.1). Literal values can be considered as resources that are not identified with URI references [50]. Formally, an RDF Graph is defined as follows:

Definition 2.18: RDF Graph [51]

Let **U**, **B**, **L** be disjoint infinite sets of URIs, blank nodes, and literals, respectively. An RDF Graph G is a four-tuple reflecting a directed labeled (cyclic) multigraph $G = (\mathcal{N}, \mathcal{L}, \Sigma, \Lambda)$ where:

- $\mathcal{N} \subset (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ a finite set of RDF resources corresponding to nodes
- $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$ a finite set of links connecting the nodes.
- Σ is a set of uniquely identified labels for the set of nodes $\mathcal{N}$
- Λ is a set of uniquely identified labels for the set of Links $\mathcal{L}$

2.3 Visual Representations

Visualizations reinforce human cognition and exploit humans' ability to understand complex data through visual representations. "*The earliest seeds of visualization arose in geometric diagrams, in tables of the positions of stars and other celestial bodies, and in the making of maps to aid in navigation and exploration*" [52, p. 3]. The advances in hardware and software engineering provide the means to analyze and show data on displays using various visualization methods. Visualizations, however, are highly dependent on the individual use case and the targeted user groups. The work of Card et al. [53] provides a definition of visualization as follows:

Definition 2.19: Visualization [53]

Visualization is the use of computer-supported, interactive, visual representations of data to amplify cognition.

As stated in Definition 2.19, the main objective of visualizations is to amplify cognition and facilitate the understanding of data. We can make a distinction for visualizations into *data visualization*, *scientific visualization*, and *information visualization*. In this thesis, we refer to visual representations as to the combination of these terms because the content of ontologies and Knowledge Graphs can express different types of data, i.e., data, scientific data, and abstract data. Data visualization refers to displaying data in a graphical fashion, where it should help in viewing the structure of the data [54]. Scientific visualization refers to visual representations of scientific data, which is often physically based. Card et al. define information visualization as follows:

Definition 2.20: Information Visualization [53]

Information Visualization is the use of computer-supported, interactive, visual representations of ***abstract*** data to amplify cognition.

Visualizations provide an abstract representation of data that reinforces human cognition. However, perception of the visualization and its underlying information plays a major role in facilitating understanding. Cognitive science shows that humans' capacity is limited in the aspect of storing and processing information [55]. Thus, with the growing size and complexity of Semantic-Web-Based knowledge structures, their graphical representations tend to become difficult to read, which is induced by visual clutter, crossing edges, and occlusion of the rendering primitives. Additionally, in graph visualizations, every rendering primitive (e.g., circle, rectangle, label, link, arrow, and even color) represents information. An information overload is a natural consequence when the amount of rendering primitives exceeds the cognitive capacity of the user to comprehend visual representation.

In human cognitive psychology, *preattentive processing* addresses an initial organization of the perceived visualization based on cognitive operations. Visual features such as color, shape, size, orientation are believed to be rapid, automatic, and spatially parallel [56]. Figure 2.6 illustrates two examples for target detection tasks. While the target detection task in Figure 2.6 a) requires minimal effort to detect the blue circle, in Figure 2.6 b) more attention is required to identify the blue circle because it does not provide a unique feature that distinguishes it from other objects in the image. In particular, the squares with identical colors in correspondence with the target object introduce additional distraction, and therefore the object detection task requires more attention.

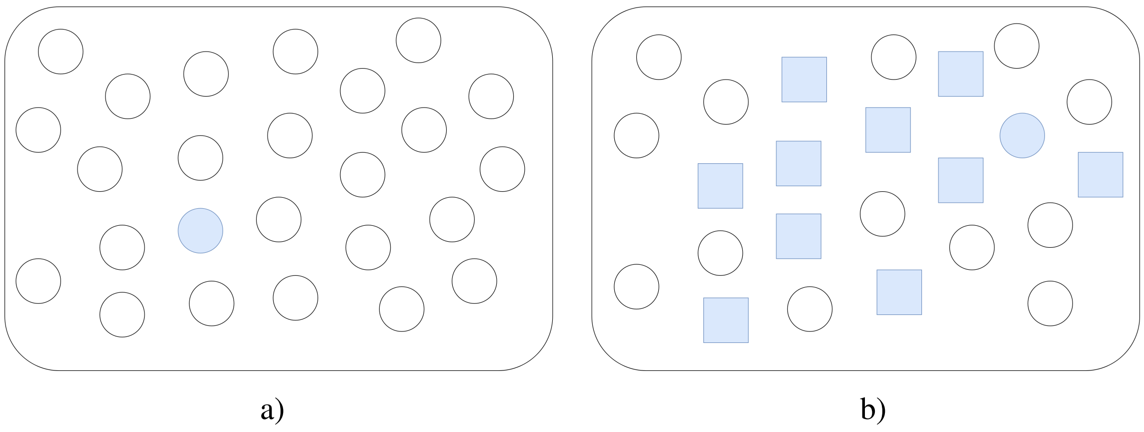


Figure 2.6: Target detection task (blue circle): a) Target element has a different color, making it easy to detect; b) Target element shares the same color with distraction elements (blue squares), requiring more attention to locate the target element. Image adapted from [56]

.

Similar effects can be observed when target objects are "masked" by other objects. This masking effect depends on elements sharing locally distinctive visual properties with other elements [57]. Figure 2.7 a) illustrates an example of such a masking effect. On a quick glance, the nine groups of objects seem equivalent. However, there are two groups with elements that are distinctive within the group. When directing attention to individual groups separately, the distinctive elements become easily observable due to the local and reduced context of the currently observed group.

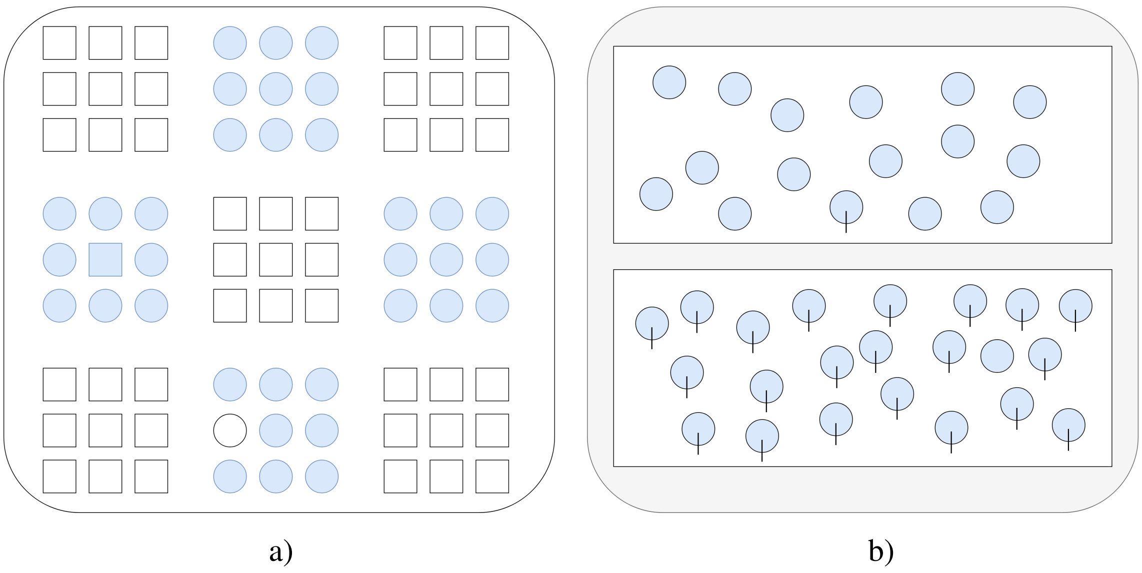

a) b)

Figure 2.7: Preattentive processing: a) Masking effect requires additional attention when elements share locally distinctive visual properties with other elements. b) Identifying the target object without a visual feature requires additional attention (bottom image). Images adapted from [57].

An additional example for target detection is illustrated in Figure 2.7 b). The upper image of Figure 2.7 b) introduces a feature (vertical line) to the target object. In contrast, in the lower image, the target detection task requires the identification of the element without this feature. Identifying the target object, i.e., the circle without the vertical line, in the lower images requires additional attention to locate the object with the absence of a feature.

A general objective of visualizations is to present data and information to humans in such a way that a visual representation is informative and meaningful. When designed in a user-centered manner, visualizations facilitate understanding of the underlying data. Additionally, visualizations should allow users to perform different types of tasks rapidly [56].

The term *mental model* is often occurring in literature and reflects the aspect of a users' internal representation of the external world. Different domains use the term *mental models* ubiquitous, but only a few provide explicit definitions [58]. Wilson and Rutherford describe mental models as the representations formed by a user for a system and/or a task that is based on previous experience and the current observation of the system [18]. In his work, Norman [59] considers four aspects for mental models: i) *target system*, ii) *conceptual model*, iii) *mental model*, and iv) *scientist's conceptualization of the mental model*. The target system reflects the conceptualization of the system; the conceptual model addresses the representation of the target system; the mental model is created by interactions with the system; and the scientist's conceptualization is a model of a mental model. Additionally, Norman observed that mental models are incomplete, unstable, do not have firm boundaries, and are unscientific. When designing visual representations for Semantic-Web-Based knowledge structures, we have to consider mental models of users as internal representations of a system addressing different aspects:

- **Interaction Layer**: Mental model of the system functionalities, e.g., interactions.
- **Conceptual Layer**: Visual representation for Semantic Web data, i.e., how does a system encode information using visual mappings. Visual notations provide examples of such mappings and formally define the depiction of data (e.g., UML notation).
- **Perception Layer**: Mental model of the perceived information, i.e., the mental model for the interpretation of the content. Ideally, the visual representation model corresponds to the user's mental model. However, these match typically *only* in some aspects and diverge from the user's expectations and previous experiences with other visualizations methods and tools.

In order to satisfy the varying demands of users and use cases, suitable visualizations require customizable representation models. Since Semantic-Web-Based knowledge structures typically reflect large networks, we require additional mechanisms to handle cognitive and information overload. The information-seeking mantra [60]: overview first, zoom and filter, then details-on-demand presents guidance for the design of visualizations. It is of interest to understand information on different abstraction levels. For example, on the general broad overview level or boiled down to specific details. Presenting all details leads to cognitive and information overload and does not support sense-making due to the aspect that such overcrowded representations are hard to understand. Users do not know where to start to create a mental model of the presented information. Thus, Shneiderman's mantra addresses it by first displaying reduced information in the form of an overview. The user then chooses the relevant information to zoom in, and details are presented on demand.

In Semantic Web contexts, visualizations often depict the structure of Semantic Web data, i.e., resources and their relations. However, more suitable visualization methods are beneficial for displaying the underlying content, e.g., chart visualizations. For example, visualizing the varying data of a temperature sensor over time using a node-link diagram results in a non-informative representation. While the nodes and links reflect the structure of the data, a line chart is more suitable to represent the content (i.e., the varying temperature over time) in a meaningful way. In summary, visualizations have to present the information on different abstraction layers with varying levels of details using suitable representations.

CHAPTER 3

Related Work

A fundamental aspect of the Semantic Web is creating and communicating conceptualizations of information and data in certain domains. Generally, Semantic Web data can be created using a text editor. However, manually creating machine-readable conceptualizations is cumbersome and can be error-prone. Furthermore, it requires the knowledge of different syntaxes prevalent in various serialization formats, such as Turtle [61] or Mancher-OWL-Syntax [62]. Modeling Semantic-Web-Based knowledge structures is becoming more and more prominent. Consequently, various ontology engineering tools have been developed in the last years. To facilitate creating and editing of ontologies, widget-based user interfaces are often used for ontology engineering (e.g., NeOn-ToolKit [63], TobBraid Composer [64], Swoop [65], OWLGrEd [66], etc.). Similar features are provided by the different tools, such as viewing the ontology structure in hierarchical trees, supporting different serialization formats, and reasoning capabilities. Popular ontology engineering tools are Protégé[1] and its descendant WebProtégé[2].

The advances in digitization create enormous amounts of data and provide new opportunities for business models. However, in order to reduce human labor and time-consuming efforts for analysis and decision-making tasks, the data has to be machine-actionable. Semantic Web technologies address such data representation formats, enabling information retrieval based on the semantics of elements (things, not strings [3]). Thus, different groups target the transformation of mere data towards information by providing structured and semantically enriched formats that are understood by machines and humans alike.

[1] https://protege.stanford.edu/
[2] https://webprotege.stanford.edu/

The work of Vu et al. [67] uses a transformation process in the form of a mapping language (D-REPR). Heterogeneous datasets, such as tables in CSV or JSON formats, with different layouts, are described in a model that defines components for the transformation into RDF. These components describe the dataset resource, its attributes, and how the data alignment is realized. Other approaches, such as XLWrap [68], focus on the transformation of spreadsheets into RDF. R2RML [69] addresses the mapping of relational databases to RDF. However, relational databases can be seen as tables, and therefore, R2RML techniques are also applied to transform tabular data into Semantic Web representations such as RDF. Most approaches use template definitions and rules for specific datasets, defining their data transformations. Due to the flexible nature of tables, the challenge of transforming tables into Semantic-Web-Based knowledge representations typically results in transformation models that are specifically tailored for individual use cases.

This chapter provides an overview of state-of-the-art approaches related to Semantic Web technologies in the context of visualization methods and tools. The remainder of this chapter is structured as follows: Section 3.1 summarizes visualization methods and tools for Semantic-Web-Based knowledge structures. Section 3.2 reviews an approach from previous work that is not part of this thesis (*Semantic Zooming for Ontology Graphs* [70]). Section 3.3 provides an overview of customizable visualization methods in Semantic Web contexts. Section 3.4 provides a summary and addresses the main research question.

3.1 Visualization Methods and Tools

Visualizations provide an abstract representation of data that reinforces human cognition. Consequently, various visualization methods and tools exist, and new ones are being developed. Figure 3.1 shows examples of existing visualization methods and tools.

Visualizations of ontologies can offer a good starting point for exploration and support the sense-making of the provided information [16]. Several ontology editors feature graph visualizations, and various visualization plugins[3] have been developed for the popular ontology editor Protégé. Nearly all of the available approaches are developed for desktop environments. Typically, they are designed to be used by specialists with sufficient knowledge in modeling and logical notations in OWL representations. However, due to the increased attention in academic and industrial contexts, various user groups are nowadays involved in ontology modeling.

[3]http://protegewiki.stanford.edu/wiki/Visualization

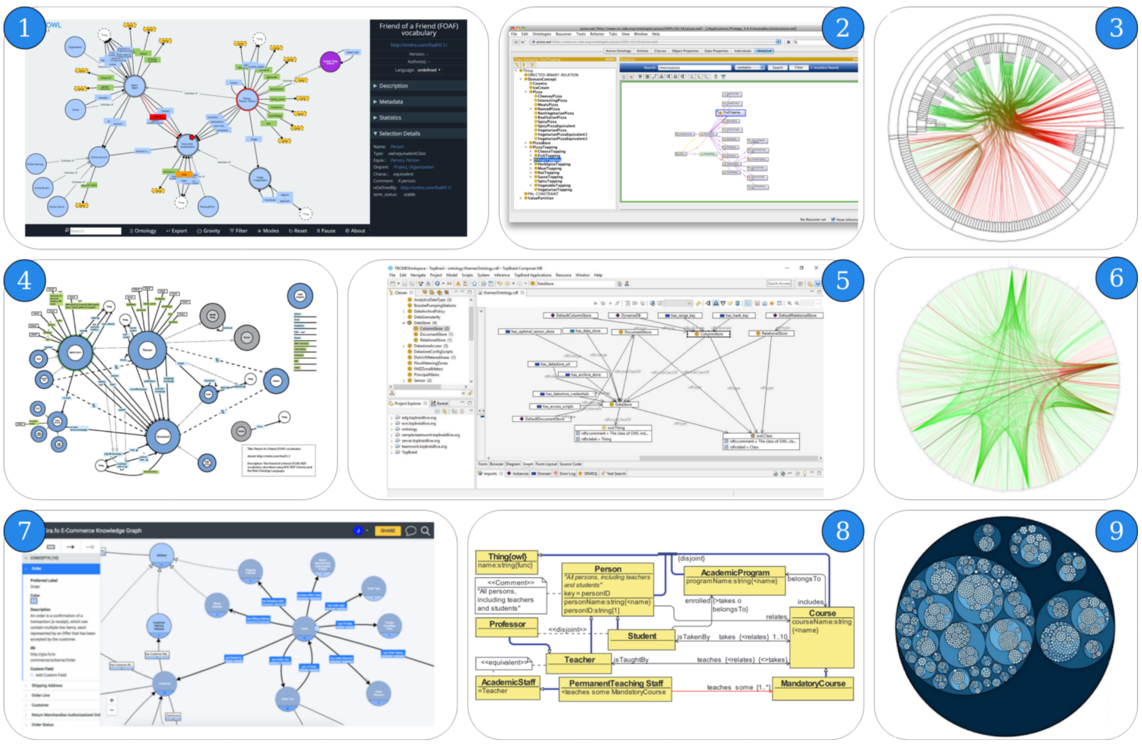
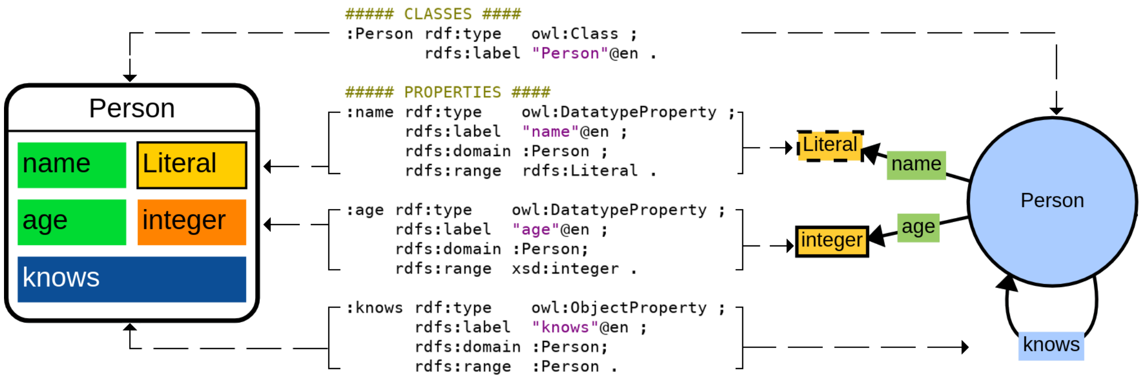


Figure 3.1: Different visualization methods and tools. ① WebVOWL, image from [71]; ② OntoGraf (Protégé plugin), image from [72]; ③ Radial tree layout, image from [17]; ④ VOWL Specification 1.0, image from [73]; ⑤ Screenshot of TopBraid Composer Maestro Edition (IDE), image from [74]; ⑥ Chord diagram in the context of Drug-Drug-Interactions; ⑦ Gra.fo, image from [75]; ⑧ OwlGred UML style graphical editor, image from [76]; ⑨ Circular treemaps, image from [77]

Dudáš et al. [17] provide a comprehensive survey of ontology visualization methods and tools. In this work, the authors identified that most approaches use two-dimensional node-link diagram visualizations to represent the underlying information in ontologies. Due to the aspect that Semantic-Web-Based knowledge representations reflect graph structures, node-link diagrams provide a well-suited graphical depiction for the underlying structure of the data. However, the various methods range from indented trees and chord diagrams to treemaps and Euler diagrams. Many ontology visualization techniques have been proposed [78–81]. Due to the aspect that different use cases, user groups, and tasks require specific visualization methods to facilitate understanding and interaction with Semantic-Web-Based knowledge structures, the majority of the Semantic Web community has not accepted a visualization method as a de facto standard [17].

Visual notations formally define the depiction of different elements for a visualization method. Examples of visual notations for ontologies are VOWL [82] and Graffoo [83], but also UML is often used to represent ontologies [84] or different tool-specific notations.

Although UML has a standard visual notation, various styles exist, such as the visual representation of ontologies with TopBraid Composer [85], a UML version of the VOWL notation [86], or the UML mapping of the NeOn Toolkit [87]. Node-link diagram visualizations are categorized into name-label-only and nested visualizations [17]: Name-label-only visualizations depict the elements of the ontology as individual labeled nodes and links. Nested visualizations (e.g., UML) aggregate information (e.g., the data properties of a class) and visualize it as a list of attributes inside the corresponding node. Figure 3.2 illustrates an example for nested and name-label-only visual representations.



```
##### CLASSES ####
:Person rdf:type   owl:Class ;
        rdfs:label "Person"@en .

##### PROPERTIES ####
:name rdf:type    owl:DatatypeProperty ;
      rdfs:label  "name"@en ;
      rdfs:domain :Person ;
      rdfs:range  rdfs:Literal .

:age rdf:type    owl:DatatypeProperty ;
     rdfs:label  "age"@en ;
     rdfs:domain :Person;
     rdfs:range  xsd:integer .

:knows rdf:type    owl:ObjectProperty ;
       rdfs:label  "knows"@en ;
       rdfs:domain :Person;
       rdfs:range  :Person .
```

Figure 3.2: Possible mappings of RDF triples to nested and name-label-only visualization.

A general aspect of visualization methods and tools is to engage different user groups and facilitate ontology exploration and modeling tasks. Nowadays, knowledge workers often use more than one device for their daily tasks in a multitude of interaction contexts, ranging from classical desktop settings to mobile scenarios in meetings, workshops, and on business trips. Thus, we observe a trend for the development of ontology engineering tools towards web-based applications. Web-based applications reduce entrance barriers to use a system due to the aspect that these are directly available, ready to use, and do not require any installation process. Examples of web-based applications in Semantic Web contexts are WebProtégé, TurtleEditor [88], OWLGrEd [66], and Gra.fo[4].

We can distinguish visualization methods and tools into the visual representation of the data structure and its underlying content. Approaches such as RelFinder [89] or the Neo4j graph visualization [90] address the visualization of Knowledge Graphs based on their structure (i.e., nodes and links). However, while node-link diagrams are well-suited to represent the data structure of Knowledge Graphs, in some contexts, such as the visualization of information encoded in tables, the structural representation will not

[4] https://gra.fo/

facilitate the comprehension of information. Knowledge Graphs have different structures and also contain additional information that does not serve the purpose of information interpretation (e.g., URIs or class assertions). Elements addressing the cell values of tables can be organized in such a way that their visual representation in the form of node-link diagrams increases the information overload due to the distant position of related values. An example for content-based visualization of Semantic-Web-Based knowledge structures is the Wikidata Query Service[5]. Users specify SPARQL queries which serve as input for different visualization methods such as Table, Tree, and Timeline for the resulting data.

Suitable visualizations are highly dependent on the individual use case, the user groups, the data at hand, and the general intent of communicating insights using a visual representation. The challenge with most approaches is grounded in their design. On the one hand, visualization methods are created with a particular definition for the representation model. On the other hand, users perceive the provided visualization and build a mental model to interpret the content [18]. Ideally, the visual representation model corresponds to the user's mental model. However, these match typically *only* in some aspects and diverge from the user's expectations and previous experiences with other visualization tools.

3.2 Semantic Zooming for Ontology Graphs

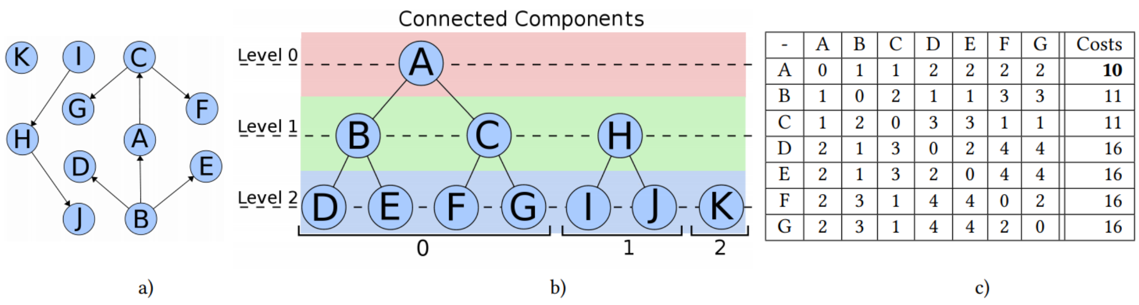
In a previous work [70], we introduced the concept of semantic zooming for ontology graph visualizations. The conceptualization of semantic zooming is inspired by the interaction mechanisms of Google Maps, where a user receives more detailed information when zooming in on a location. In the zoomed-out state, the user views an abstract and reduced representation of information. While this conceptualization is intuitive for 2D map representations, it is not intuitive to apply it to graph visualizations. Due to the aspect that graph data does not provide spatial arrangements its is not obvious how to design different information zoom levels.

In this work, we analyzed the underlying graph structure and built connected components for the graph. We compute a minimum spanning tree for each connected component, which is based on its topological structure. Thus, each connected component provides a root node from which a user can start exploration tasks. We construct the minimum spanning tree by converting the directed multigraph of a connected component into a simple undirected graph. Within this simple graph, we compute for each node a cost measurement. For a

[5]`https://query.wikidata.org/`

node n_i, the minimal distance to a node n_j is determined by the number of links required to traverse from node n_i to n_j. The cost measurement is the sum of all distances from node n_i to any other node within the connected component. The node with the minimum costs is selected as the root node of the minimum spanning tree of a connected component. When identical minimal cost measurements exist, multiple nodes are selected as root nodes.

Using the assigned depth of nodes in the minimum spanning tree with minimal costs allows us to define a hierarchy for the exploration of the nodes. Furthermore, the transformation of parallel links into a simple undirected link allows us to reduce the visual clutter when numerous connection between two nodes exists. We call this transformation *link-collapsing* and its inverse *link-expanding*. Similarly, the hierarchy of the minimum spanning tree allows us to reduce the number of visualized nodes in the graph using *node-collapsing*. The node-collapsing operation hides the sub-tree attached to a particular node. In contrast, the *node-expanding* operation shows all nodes on the next depth level in the hierarchy connected to a specific node.



-	A	B	C	D	E	F	G	Costs
A	0	1	1	2	2	2	2	**10**
B	1	0	2	1	1	3	3	11
C	1	2	0	3	3	1	1	11
D	2	1	3	0	2	4	4	16
E	2	1	3	2	0	4	4	16
F	2	3	1	4	4	0	2	16
G	2	3	1	4	4	2	0	16

Figure 3.3: Assignment of the global topological levels of detail: a) Input graph, b) Minimum spanning tree organization, c) Path matrix for the largest connected component of the input graph, indicating the computation for the exploration costs.

The semantic zooming approach for ontology graph visualizations uses three different information layers: **i)** Topological Layer, **ii)** Aggregation Layer, and **iii)** Visual Appearance Layer. The simplification and abstraction of the visual representation are reflected in the form of discrete levels of detail for each information layer, which allows the user to focus on certain regions in the graph and explore its structure. The topological layer addresses the exploration of nodes using their hierarchy which is computed using minimum spanning trees for individual connected components of the graph. The aggregation layer addresses the exploration of relations between nodes using link-collapse and link-expand operations. Furthermore, we define self-reflexive properties (self-loops) and datatype properties and their asserted literals as attributes of a node. Thus, the aggregation layer provides the means to investigate the attributes of individual nodes. The Visual Appearance Layer addresses how rendering primitives are displayed to the user depending on the geometric zoom level in the graph. Similar to Google Maps' geometric zoom levels, we provide more details when zoomed into a graph and reduce the details when zoomed out. We observe that the class names and additional symbols become too small to read on a zoomed-out visualization of the graph. No further insight is obtained when the user can not clearly see or read the provided rendering primitive. Instead, these small primitives introduce noise and distraction to the visualization. Thus, excluding these unreadable rendering primitives results in a much clearer visualization, which additionally boosts the performance, as fewer primitives stress the rendering pipeline. We obtained the definitions of the discrete detail levels for the Visual Appearance Layer layer from expert interviews with five participants who had at least five years of experience in the ontology domain. Figure 3.4 shows the defined visual representations for different geometric zoom levels.

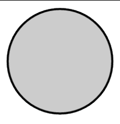
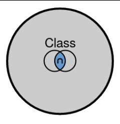
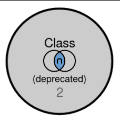
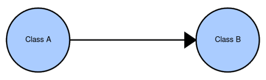
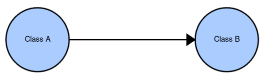
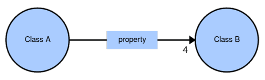
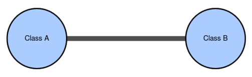
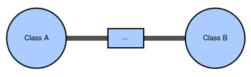
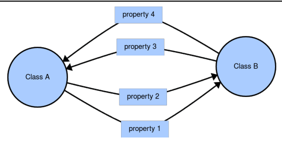
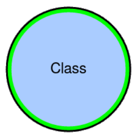
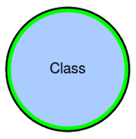
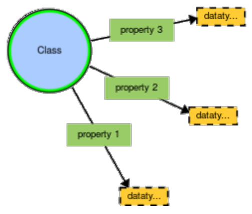
Geometric Zoom	**0.25**	**0.5**	**1.0; 2.0**
Classes			
Links			
Multi-links			
Datatypes and Self-Loops			

Figure 3.4: Discrete level of details defined by expert interviews.

A user study was conducted in order to evaluate the benefits of ontology graph visualizations enhanced with semantic zooming. The user study included 12 participants. The participants comprised computer science students of the University of Bonn and employees of the Fraunhofer IAIS institute. The age of the participants was in the range of 23 and 63 years, with a median of 29 years. The user study results reveal that the application of ontology graph visualizations enhanced with semantic zooming outperforms the one without it. This is reflected in the higher ratings for evaluated metrics (i.e., *Readability*, *Visual Clarity*, *Information Clarity*, *Navigation Support*, and *Layout Stability*).

With the growing size and complexity, the corresponding graph visualizations exhibit visual clutter, information overload, and occlusion problems, making the underlying structure of the ontologies challenging to comprehend. Our approach partitions and organizes the information of an ontology into three information layers with discrete levels of detail, which allow the user to control the amount of the presented information. The results and conceptualizations of this work are used as foundations for later work to mitigate cognitive and information overload for the visualization of Semantic-Web-Based knowledge structures. In particular, we apply node-link collapsing and expanding interactions for the visual exploration of such data structures.

3.3 Customizable Visual Representations

Visualizations are highly dependent on the individual use case and targeted user groups. However, a visual representation model has to correspond to the user's mental model in order to provide a suitable visualization. Thus, only flexible and customizable visualization approaches for ontologies can fulfill the demands of different use cases and user groups. The number of visualization methods, tailor-suited tools, along with the requirements and necessity for customization, indicate that a *one-size-fits-all* solution is challenging, if not impossible, nor feasible, to realize.

Early work by Pietriga et al. [91] develops the concept of Fresnel Lenses, a presentation vocabulary for RDF. Lenses, formats, and CSS classes are responsible for the visualization of RDF data. The objective of the lenses is to select the content and apply custom orderings of the data. The formats and the CSS classes define how the information is presented. IsaViz [92] is a related approach that enables the definition of visual representations for ontologies based on Graph Style Sheets (GSS) [93]. GSS are similar to CSS and use a selector to which attributes are assigned. Cytoscape [94] is a visualization tool that applies a similar approach in order to enable customizable visualizations of node-link diagrams.

GSS and CSS enable the definition of styles for rendering primitives. However, they do not address the spatial positioning as required in graph-based visualization methods. Furthermore, GSS and CSS do not operate on OWL constructs but apply styles to elements in the DOM tree. Also, specific requirements, such as the distinction between name-label-only and nested visualizations are not supported by these languages. Thus, GSS and CSS lack the capabilities required for the comprehensive representation of graph-based ontology visualizations and are not sufficient in this context.

The Web Annotation Data Model [95] defines a model for describing associations between resources. In this model, annotations define a target and a body, where the body contains additional data that should be associated with the target resource. It provides a standard description method for annotations to be shared between systems. While allowing for the definition of style information as annotations, it is designed for the general annotation of resources and not for the visual representation of ontologies.

3.4 Summary

Different methods and tools exist for inspecting and creating Semantic-Web-Based knowledge structures. Some initiatives address the creation and population of Knowledge Graphs from heterogeneous datasets [67, 96]. Others address the quality of the underlying information, for example, by integrating new explicit knowledge using link prediction [97].

Semantic Web technologies are typically designed for machine-readable and machine-actionable representation of knowledge. However, visualizing the underlying information plays a major role in communicating the insights to people. Thus, different visualization methods and tools have been created in the last years, and new ones are being developed.

This thesis studies how to facilitate understanding of Semantic-Web-Based knowledge structures through visual representations. Research question 1 (**RQ1**) addresses how to facilitate and engage various user groups with different backgrounds in ontology modeling using a device-independent visual modeling approach. Visual modeling approaches reduce the complexity of OWL formalizations to create machine-readable representations of knowledge. While Semantic-Web-Based knowledge structures are often displayed as node-link diagrams, their visual appearance and spatial layout may diverge from users' mental models. Thus, research question 2 (**RQ2**) addresses customizable visual representations for realizing suitable visualizations for different use cases and diverse user groups. The varying requirements of different use cases and user groups often result in tailor-suited solutions which are not always extendable or reusable. Thus, research question 3 (**RQ3**) addresses how to facilitate the creation of visual representations and enable their reuse for different use cases and user groups.

CHAPTER 4

Semantic-Web-Based Knowledge Structures from a User Perspective

A fundamental aspect of the Semantic Web is to *create* and communicate conceptualizations of information and data in specific domains. In this chapter, we address the creation and editing process of Semantic Web-Based knowledge structures from the ontology development perspective. Ontology modeling typically involves ontology engineers and domain experts with different backgrounds in Semantic Web technologies. On the one hand, ontology engineers typically lack the expertise in a certain domain to provide ontologies of sufficient quality [15]. On the other hand, domain experts are often not familiar with conceptual modeling techniques and find it hard to follow logical notations in OWL representation. Visualizations in the form of node-link diagrams are commonly used to support ontology modeling and related tasks. These types of visualizations allow for displaying the structure and the interrelations between elements of ontologies. Involving domain experts more directly in ontology modeling requires user-centered and immediately available approaches that are easy to use and independent of the device and interaction context. This chapter addresses the following research question:

Research Question 1 (RQ1)

How can we ease the creation and editing process of Semantic-Web-Based knowledge structures from a user perspective?

The contributions of this chapter are the following:

- Requirements for a device-independent visual modeling approach.
- A preliminary evaluation, comparing our application, WebProtégé, and TurtleEditor.
- A comparative user evaluation on visual ontology modeling using node-link diagrams, comparing our visual modeling application to Protégé.

This chapter is based on the following publications: [98, 99][1]

The remainder of this chapter is structured as follows: Section 4.1 presents a device-independent visual modeling approach. In Section 4.2, we present a comparative user evaluation on visual ontology modeling using node-link diagrams. Finally, in Section 4.3, we summarize the achieved results in relation to the research question **RQ1**.

4.1 Device-Independent Visual Modeling

Knowledge modeling is often done collaboratively in joint efforts of knowledge engineers and domain experts. However, domain experts are typically not familiar with semantic formalism and conceptual modeling techniques. Approaches that reduce the complexity of creating and editing Semantic-Web-Based knowledge structures are necessary to facilitate the joint efforts of ontology engineers and domain experts.

Visualizations of ontologies can offer a good starting point for exploration and support understanding of the provided information [16]. In particular, node-link diagram visualizations are often used to depict the structure of ontologies [17, 82]. Numerous ontology visualization techniques have been proposed in the last years [78–80]. The work of Dudáš et al. [17] provides an overview of different visualization methods and tools for ontologies.

Several ontology editors feature graph visualizations, and various visualization plug-ins have been developed for the popular ontology editor Protégé[2]. Nearly all of the available approaches are developed for desktop environments, and only a few of them support direct editing within the visual representation [78]. Furthermore, most ontology modeling tools need to be downloaded and installed first, which increases the entry barriers for using the software. Installation processes, as easy as they have become nowadays, often require administrator privileges, which may not be accessible to all users, especially in enterprise contexts. In contrast, web-based applications are immediately available and ready to use.

[1] [99] is a joint work with Rohan Asmat, a former colleague from Fraunhofer IAIS (Enterprise Information Systems). I contributed to developing the study design and the presentation of the obtained results.

[2] `http://protegewiki.stanford.edu/wiki/Visualization`

While existing web-based ontology editors, such as WebProtégé [100] or TurtleEditor [88], can principally be used with touch devices, they are mainly designed for the classical WIMP (windows, icons, menus, pointer) interaction paradigm. We present an approach that addresses the visual modeling of ontologies by defining interaction modes for different devices. An approach related to ours is OntoSketch [101], which allows the users to model ontologies on a tablet using pen and sketch interactions. However, it is designed for a particular interaction scenario and type of device (i.e., sketching on a tablet) but does not address different interaction contexts and devices. Similar to related approaches, we provide built-in constraints to guide the users towards best practices in ontology modeling. We are not aware of any other work that applies device-independent interaction modes for visual modeling of ontologies.

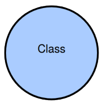
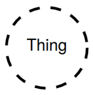
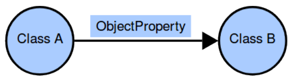
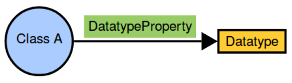
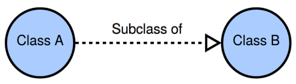
Device-independence is becoming more and more important in ontology modeling for different reasons: Diverse types of computing devices, such as tablets, smartphones, convertibles, and touchscreens, are increasingly used in work environments. Nowadays, knowledge workers often use more than one device for their daily tasks in a multitude of interaction contexts, ranging from classical desktop settings to mobile scenarios in meetings, workshops, and on business trips. Also, digital devices for creativity techniques, such as interactive whiteboards, idea walls, and touch tables, are increasingly available and used for idea generation and conceptual modeling. Therefore, we aim for an ontology modeling approach that is immediately available, intuitive, easy to use, and independent of a particular device and interaction context and exploits novel sensors and interaction techniques. In numerous industrial engagements (e.g., [102]), we observed that visual modeling is a top-requested feature by industrial users and a prerequisite for wide penetration of Semantic Web technologies in general. Only by providing visual modeling techniques as they are prevalent with other modeling techniques such as UML, BPMN, or ARIS, Semantic Web technologies will be able to gain further industrial importance.

We present an approach based on the VOWL notation, a well-specified visual language for the user-oriented representation of OWL ontologies. VOWL aims for an intuitive and comprehensive representation that is also understandable to users less familiar with ontologies [82]. The node-link diagram represents the ontology concepts as a graph $G(N,L)$. Using the terms of the ontology, the set of nodes N reflects its classes that are connected via properties by the set of links L. Table 4.1 shows a selection of visual elements in the VOWL specification[3], indicating the visual mappings for nodes and links.

[3]The VOWL specification is available at: `http://purl.org/vowl/spec/`

By implementing the approach with standard web technologies and taking touch interaction into account, we achieve a high device-independence and remove the need for users to install any software. Our implementation supports both mouse and touch operation, providing a similar interaction behavior on different devices. Additionally, we integrate a set of built-in functions avoiding common pitfalls and preventing invalidation of ontologies.

Table 4.1: Subset of the VOWL specification for the visual representation of OWL.

Ontology element	Graphical representation	Description
`owl:Class`		Circle showing the class label in the selected language.
`owl:Thing`		This representation is multiplied in the visualization according to the splitting rules.
`owl:ObjectProperty`		Representation of an object property, showing the property label in a rectangle.
`owl:DatatypeProperty`		Representation of a datatype property with multiplied datatype nodes according to the splitting rules.
`rdfs:subClassOf`		This representation is similar to the UML notation for generalizations.

4.1.1 Requirements

In the following, we summarize the main requirements we have identified and followed in developing our approach for device-independent visual modeling of ontologies to foster a more direct engagement of domain experts. The requirements were derived from sessions with domain experts in the context of the research project GRACeFUL[4], and in industry workshops conducted with domain experts from the manufacturing and healthcare domains.

[4]https://www.graceful-project.eu

Easy to Understand and Learn — The first and most important requirement is that the modeling approach is user-centered and designed to serve the skills and needs of domain experts with limited knowledge of ontology engineering and the OWL modeling language. This requirement is addressed using the VOWL notation, targeting ontology representations in the form of node-link diagrams that are easy to understand [82]. Visual editing fosters engagement and makes the approach easy to learn using built-in constraints for modeling in compliance with the OWL specifications, best practices, and conventions.

Easy to Access and Use — Web-based ontology editors, such as WebProtégé [100] or TurtleEditor [88], are immediately available as they do not need any software installation and configuration. They can be used from anywhere, as long as a web browser is installed on the operating device. The usability is further increased if the user can continue modeling while being offline, i.e., when not permanently connected to the internet.

Independent of Device and Interaction Context — Available ontology modeling approaches, including web-based ontology editors, are not yet fully device-independent. They are still mainly designed for classical WIMP (windows, icons, menus, pointer) interaction modes. Therefore, often neglecting other device and interaction contexts, such as touch operation and varying display sizes. However, we consider the easy access and operation of the modeling software to be a crucial aspect of lowering the entry barriers and engaging more user groups in ontology modeling, independent of the operating device and interaction context.

Guidance — We have identified guidance during the modeling and editing process as an important requirement. A visual modeling approach based on node-link diagrams allows for flexible editing with many degrees of freedom. Neglecting any restrictions during the modeling process can quickly result in invalid ontologies since domain experts are often little familiar with the OWL ontology language's possibilities and constraints. Thus, they need to be guided, for instance, with regard to the elements they can connect in the visualization or the text they can enter. This way, guidance mechanisms can even help to train unfamiliar users towards best practices in ontology engineering.

4.1.2 Device-Independence

The variety of available devices demands several types of input interactions for the task of ontology modeling. In sessions with domain experts (cf. Section 4.1.1), we identified the following devices and modes of operation to be most likely used in ontology modeling:

1. PC: keyboard, mouse, monitor
2. Laptop: keyboard, mouse, touch-pad, monitor
3. Touch-operated devices such as tablets, smartphones, and touch displays

A device-independent approach for ontology modeling has to cover all the above-defined devices and enable different modes of operation that are shown in Table 4.2.

Table 4.2: Device-related input interactions and their classification for possible usage.

Input Device	Possible use for visual modeling	Device-independent
Keyboard	Shortcuts & keyboard modifiers to create and edit elements	No, touch devices provide a display keyboard, but typically they do not have keyboard modifiers like “alt” or ”control”
Mouse	Point, click, and drag interactions	Yes, touch interactions are similar (click replaced by touch)
Touch displays and touch pads	Touch and drag interactions	Yes, similar to point, click, and drag interactions
Integrated cameras	Gesture detection	No, not all devices have integrated cameras (e.g., PC) and would require additional hardware
Integrated microphone	Speech recognition for creating elements and their interrelations	No, not all devices have integrated microphones (e.g., PC) and would require additional hardware
Acceleration sensors	Navigation inside the graph	No, not available on all devices

Separate deployment methods can be realized for target devices. However, tailor-suiting or even creating apps for tablets or smartphones includes additional development effort, which is not always desirable or maintainable by the developers. Additionally, the installation process implies entry barriers for using the software, as previously stated in Section 4.1.1. Administrative rights may not be accessible to all users, particularly in enterprise contexts. Requesting permission to install software can pose a hard constraint on the usage of a software tool.

Web-based approaches neither require any installation process nor any administrative privileges. Web-based applications can run directly in web browsers on different devices with different interaction contexts. Therefore, any device-related entry barriers are reduced, i.e., the software is easy to access and ready to use. However, web applications are not sufficient to achieve *full* independence from the defined set of devices. Different modes of operation imply additional restrictions. For example, a "mouse right-click" or keyboard modifiers like "control" or "shift" are not available on touch devices. We address these restrictions by defining minimal interaction methods with the visual modeling frame for both touch and mouse-operated devices. The defined modes of operation are separated into *creation* and *editing* actions and are listed in Table 4.3 and Table 4.4, respectively.

Table 4.3: Interaction methods for **creating** elements.

Creation	Mouse	Touch
Class	Double click on free canvas	Double tap on free canvas
Object property	Hover, drag & drop arrow-head	Select, drag & drop arrow-head
Datatype property	Hover & click button (+)	Select & tap button (+)

Table 4.4: Interaction methods for **editing** elements.

Editing	Mouse	Touch	Alternative
Removal	Hover & click button (x)	Select & tap button (x)	None
Class name	Double click on class	Double tap on class	Input in sidebar
Property name	Double click on property	Double tap on property	Input in sidebar
URI	Input in sidebar		
Class type	Dropdown selection in sidebar		
Property type	Dropdown selection in sidebar		

4.1.3 Visual Modeling

The reduction of device-related entry barriers addresses the usability of the approach. However, the complexity of an approach and its representation of the comprised information play crucial aspects to engage different user groups in ontology modeling. Several works address exploration, verification, and sense-making of information in ontologies by transferring the machine-readable textual representation into visual notations, such as node-link diagrams [103, 104], adjacency matrices [105], or UML-like visual representations [66].

Visualizations provide an abstract representation of the comprised information of an ontology, allowing for viewing its structure and the interrelations of different elements. Thus, visualizations reduce the complexity of textual formalization of ontologies. Visual modeling provides an abstract method for creating and editing domain knowledge conceptualizations using direct manipulations in the visual representation. However, most of the approaches require an understanding of a relatively complex visual notation (e.g., UML). We address the different backgrounds of domain experts and other user groups using the VOWL notation that is easy to understand.

Visual modeling is realized by minimalistic device-independent modes of operation for mouse and touch interactions. Regardless of the reduced complexity in ontology modeling using direct interactions within the visualization, OWL specifications and logical formalizations are hard to follow for the domain experts. Thus, we reduce possible OWL constructors to basic elements for a better user experience during the ontology modeling process. We define a *top-down* approach for ontology modeling to balance between a simple conceptualization and a fully matured ontology. An optional refinement process follows the creation of general elements.

The VOWL notation allows the visualization of the T-Box of an ontology. Its combinations of visual elements (e.g., `rdfs:Literal` and `owl:Thing`) provide for a simplified conceptual view of the T-Box. The T-Box of an ontology can be seen as a set of conceptualizations (classes) and their attributes. The attributes represent self-reflexive relations, interrelations between the classes (object properties), and their corresponding datatype properties. Using this simplified abstraction, the basic constructors in our visual modeling approach are `owl:Class`, `owl:ObjectProperty`, and `owl:DatatypeProperty`.

Nodes in the VOWL notation represent ontology classes (`owl:Class`). Nodes are created by performing a double click or double tap interaction on the empty space of the canvas area, see Table 4.3. In the VOWL notation, class-attributes, e.g., self-loop properties, object properties, and datatype properties, are assigned to one particular node. The class-attributes are created and modified through visual modeling elements (buttons, arrow-heads, and arrow-tails) directly within the visual representation.

To reduce visual clutter, these visual modeling elements are presented to the user when an editing operation of a particular element is desired. For device-independence, we define that the desire to do so arises when the user selects an element. Additionally, on point and click devices, we show the visual modeling elements on mouse hovering to reduce the interaction to first selecting an element.

The interrelations, in the form of links, between nodes correspond to object properties in OWL. Links are created by dragging the arrow-head from a source to a target node, defining the domain and range restrictions of a corresponding `owl:ObjectProperty`, respectively (cf. Table 4.3). Self-loop properties are created by dragging the arrow-head into the source node.

Datatype properties are created by pressing the (+) button of a particular node (cf. Table 4.3). In the VOWL notation, datatype properties are always connected to a multiplied visual datatype element. Thus, this action creates an `owl:DatatypeProperty`, where the domain is the class of operation, and the range is `rdfs:Literal`. Additionally, our approach supports a set of typed literals such as `xsd:integer` and `xsd:boolean`.

The editing of class and property names (`rdfs:label`) is realized directly in the graphical representation of the corresponding element. In contrast, other specifications (e.g., URI's or different types for classes, datatypes, or properties) are realized using text input or dropdown selection in a collapsible sidebar panel only, see Table 4.4. Additional refinements for classes, datatype properties, and object properties are realized by a set of checkboxes with respect to the type of an element.

Guidance and additional built-in constraints are described in the following section, which addresses the implementation[5] details of the approach.

Usage and Implementation Details

The proposed approach is implemented by extending the existing web-based framework WebVOWL. Additional modifications are made to realize the different output and input modalities of the defined set of devices. As output modality for the devices, we consider a display and its corresponding size. Thus, the viewport-settings for rendering the web page are defined as "initial-scale=1.0","maximum-scale=1.0" and "user-scalable=0". These settings prevent the user on a touch device to accidentally zoom on the web page, where the actual intention is to zoom on the graphical representation area. Additionally, the responsive design automatically optimizes the web page layout, depending on the display's size and orientation. For the input modality on touch devices, the double-tap operation is replaced. Typically, the double-tap operation performs zooming on the web page.

In our implementation, the zooming operation can be performed using the multi-touch operation (two-finger zooming) and alternatively using the slider (for a single touch device,

[5]The current implementation of WebVOWL with visual modeling functionality is available at: `http://editor.visualdataweb.org/`

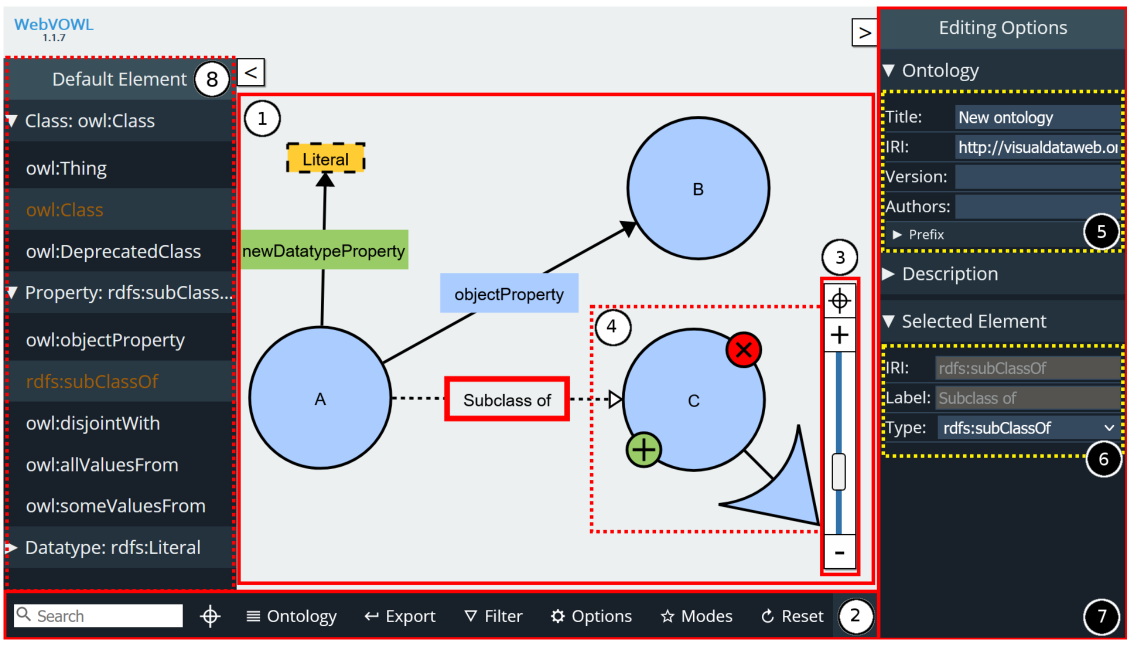


Figure 4.1: Overview of the user interface: 1) Canvas area for the visualization and direct modeling; 2) Menu with controls; 3) Zooming slider with the possibility to locate and zoom to the center of the visualization; 4) Editing elements displayed for a hovered class; 5) Ontology annotations; 6) Details for a selected element (here, “Subclass of”). 7) Collapsible sidebar panel, providing editing functionality. 8) Default OWL construct selections for creation functionality.

like a touchpad on a laptop). The slider is provided in the lower right corner of the canvas area, see Figure 4.1 ③. Additionally, an automatic smooth zooming functionality is implemented based on the approach of van Wijk [106]. This automatic zooming function determines the center position of a bounding box for all visual elements. Based on the display size, this function computes a target zooming factor. The target zooming factor and the center position determine the parameters for a smooth zooming and transition animation. This function is accessible by the target button, located at the top of the slider control elements. Thus, zooming through a double-tap becomes redundant, allowing us to use this operation for visual modeling, particularly for the creation of nodes representing the classes of an ontology.

The creation of interrelations between the nodes is realized through a drag-able arrow-head. The size of this particular visual element is chosen w.r.t to the occlusion of the finger. Additionally, the in-accuracy of touch devices was considered in the implementation. A connection is generated when the arrow-head is placed near a node. Restrictions on datatype nodes prevent incorrect assignment of object properties to datatypes. Furthermore, datatypes are characterized as leaf nodes without interlinking capabilities.

The basic constructors create the visual elements, a default URI, and an `rdfs:label` for each element. The default URI is constructed from the ontology URI, the type, and the instance id of the element (e.g. `.../Class0`, `.../objectProperty0`, or `.../datatypeProperty1`, where classes and datatypes have a different instance counter compared to object properties and datatype properties). The default label is set in a similar fashion to "NewClass", "newObjectProperty", or "newDatatypeProperty", based on the corresponding type of the visual element.

The editing for the labels of classes and properties is supported using visual modeling directly inside the graphical representation, see Table 4.4. This visual modeling interaction focuses and selects the label, allowing a direct replacement of the whole text without the need of first deleting the corresponding characters. Text input elements in the sidebar provide an additional alternative to modify the labels. Furthermore, devices with speech recognition can use it as textual input functionality.

The elements' URI modification does not have a visual modeling functionality and can only be edited in the sidebar. Additional characteristics for classes and properties (e.g., symmetric, transitive, etc.) are provided in the sidebar on selecting an element. The modification for a type of an element is realized using dropdown selection also in the sidebar. Thus, the collapsible sidebar provides refinement capabilities intended to be used by ontology engineers with the corresponding knowledge about the OWL specification.

The implementation of the current approach realizes basic guidance functionality. Two aspects are considered here for guidance: The guidance towards best practices in ontology development (creation and publication) and the guidance in the visual modeling itself. While the latter is realized through the VOWL notation and built-in constraints, e.g., a self-loop property excludes `rdfs:subClassOf` for type selection in the sidebar. Additional pop-up messages guide the user through the modeling process. For example, a warning is displayed when a delete operation will remove three or more visual elements. The guidance towards best practices in ontology development is more challenging.

The current approach's implementation provides a set of annotation properties for the ontology itself, i.e., title, version, authors, and description. All annotations and labels are currently defined in the English language (i.e., using the suffix `@en` for literal values). The envisioned multilingual support should allow the creation in multiple languages and, as a feature for future implementation, warn the user when not all elements have defined labels or annotations in a set of languages for the modeled ontology. Additionally, we enforce a naming convention. Classes start with a capital letter, while properties have a lower-case starting letter. This naming convention is applied for a class or property name based on the element type. An exception is made for `owl:Thing` and `rdfs:subClassOf`. The URI is automatically set to the corresponding element, and the labels are set to "Thing" and "Subclass of", respectively. Furthermore, the editing functionality for the URI and the label is disabled.

4.1.4 Preliminary Evaluation

A preliminary evaluation has been conducted for the implemented approach. In this study, we compare our approach with two other web-based tools that also aim to lower the entry barrier in ontology modeling and can potentially be used on different devices: **i**) WebProtégé[6], and **ii**) TurtleEditor[7]. Our preliminary evaluation consists of six participants with different backgrounds in semantic technologies and ontology modeling. Participation was voluntary and without any payment. The participants consist of a psychology student of the University of Bonn and Fraunhofer IAIS institute employees. The participants' age was in the range of 25 and 64 years, with a median of 35 years. In this study, a classification into non-expert, intermediate, and expert users was performed based on the answers addressing their experience with visual modeling and ontologies. Participants with experience in visual modeling and ontologies were classified as expert users, whereas having expertise in only one area leads to a classification as an intermediate user. A non-expert user, in this classification, has neither experience in visual modeling nor ontology development. All experiments have been performed on a `Samsung Galaxy Tab 2 GT-P5110` tablet in order to establish a controlled environment.

Study Design

The first part of the study collects information about the participant's gender, age, and experience with visual modeling and ontologies, followed by task descriptions of the study. The participants were asked to create a simple conceptualization of a family based on a given set of concepts (Person, Father, Mother, Child, Son, and Daughter) and define how these are related to each other. The process of conceptualization and thought organization could be supported by pen and paper, which all participants used in order to draw sketches similar to node-link diagrams, as shown in Figure 4.2.

[6]WebProtégé: `https://webprotege.stanford.edu/`
[7]TurtleEditor: `http://editor.visualdataweb.org/turtle-editor.html`

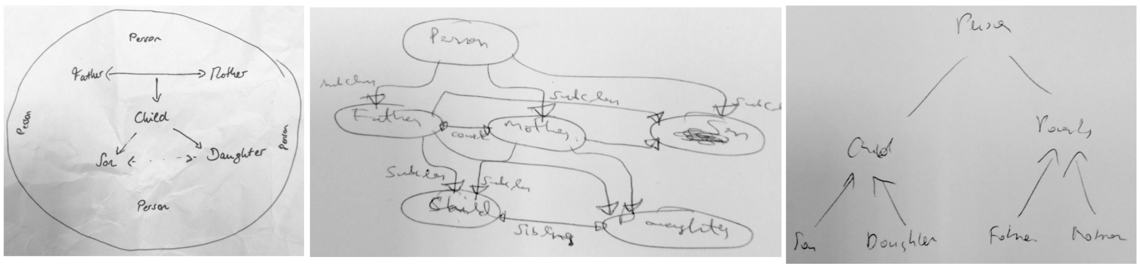


Figure 4.2: Sketches for the conceptualization of a family. From left to right: sketches of a representative participant from non-expert, intermediate, and expert user, respectively.

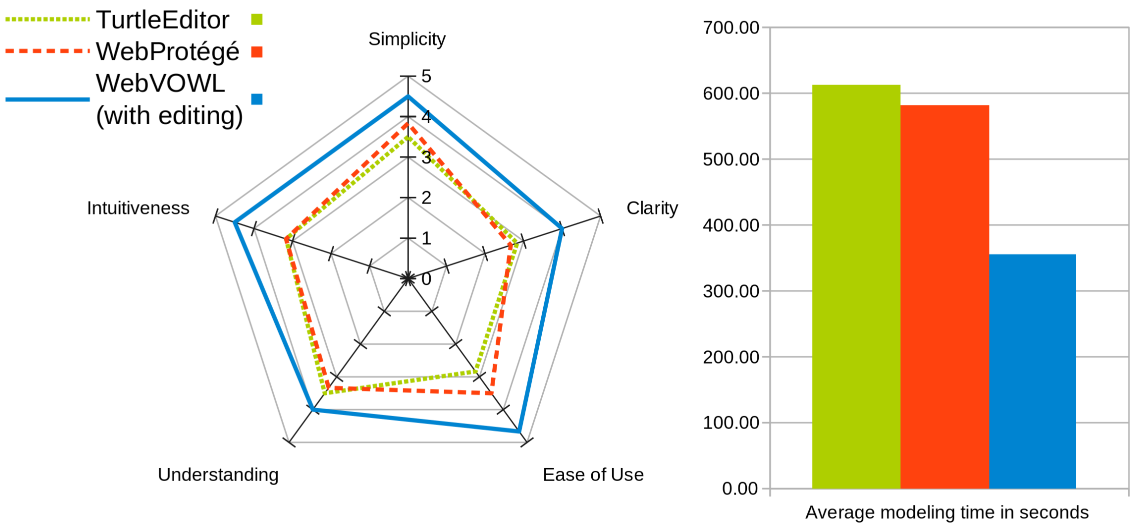


Figure 4.3: Star plot showing the obtained user ratings for the different tools used in the modeling process. A bar chart illustrates the average time required for the modeling task for the corresponding tools, i.e., TurtleEditor, WebProtégé, and WebVOWL Editor.

Three consecutive experiments were performed after the conceptualization of a family was defined. The task of modeling an ontology was addressed in each experiment using different tools. Additionally, we recorded the required time for the ontology modeling process for the individual tools, respectively. An assessment, where the participants had to rate the different tools along various dimensions, was performed directly after each experiment. The rating on each dimension used a scale ranging from one to five, where one maps to very low and five maps to very high. After the rating, the participants were asked to list positive and negative aspects and provide suggestions for improving the corresponding tool in an open discussion.

Results and Discussion

The preliminary user study results indicate the additional benefits of the proposed approach for device-independent visual ontology modeling. This is reflected in the higher average scores, as shown in Figure 4.3, for *Simplicity*, *Intuitiveness*, *Understanding*, *Ease of Use*, and *Clarity*. Additionally, the required time for the modeling task is on average reduced. In order to consider the learning effect in the modeling task, we performed a counterbalancing approach by alternating the order of tools for each participant. Nevertheless, these study results can only be considered as preliminary with respect to the small group of users and the evaluation on a single device, i.e., a tablet.

In this study, we observed that, when it comes to conceptualization, all users draw sketches in the form of node-link diagrams to organize their thoughts. Thus, visual modeling approaches in the form of node-link diagrams are suited for the development of ontologies. After the participants performed the modeling task with WebVOWL Editor and then continued with the next approach, an additional observation occurred. The participants transferred the double-tap interactions to create a node or edit a label to the other approaches that did not support this functionality. This emphasizes that the chosen interactions are intuitive and easy to learn. Nevertheless, a short introduction to the interaction methods, particularly how to create a node, was necessary. Some users negatively remarked on the missing explanation. Their suggestion for further experiments was to provide an introductory tutorial for the individual tools. A surprising observation in this study was that one of the non-expert users, having no prior experience with visual modeling nor ontologies, out-performed even the experts using WebProtégé. This emphasizes again how important approaches are that not only consider the input and output modalities of the different devices but also address different backgrounds of the participants using them. This particular participant commented on this outstanding result as “Easy hierarchical view and generation, which is missing in both other tools.”. Participants, who made an error by creating “Mother” as a subclass of “Father”, tried to realign the tree visually or searched and tried different interactions. This visual realignment of concepts is supported by WebProtégé, but only on point and click devices. On a touchable device, the layout of the web page is not optimized for the display size. Thus, trying to perform a dragging operation on a concept within the hierarchical tree results in the translation of the web page. Some of the participants negatively remarked on this non-optimization of the layout. Additionally, one aspect raised by two participants was that a generation of multi-hierarchy conceptualizations is not possible using this hierarchical tree approach.

The visual modeling using TurtleEditor was negatively remarked as not accurate enough for touch devices. This is explained by the small visual modeling canvas and the occlusion problems of the finger. The participants experienced several times that a link was not generated even if they assumed that they dragged the link to the node of interest. Nevertheless, the means of visual modeling are positively remarked in both visual approaches (TurtleEditor and WebVOWL with editing functionality).

The participants with prior experience in ontology modeling emphasized that this approach could be useful in ontology development and could involve the domain experts in the modeling process. Some participants characterized the force-directed layout in WebVOWL, providing a dynamic reorganization of the visual elements as “fun to use”. However, other participants found it distracting and paused the force-directed layout during the modeling process. This observation emphasizes once more that different user groups have divergent preferences to how they interact with the modeling environment.

4.1.5 Summary of the Approach

In this section, we presented an approach for device-independent visual modeling of ontologies that lowers the entry barriers for domain experts to get more directly involved in ontology modeling. Most of the available approaches are designed to be used by expert users with sufficient training in modeling and logical notations in OWL representations. Additionally, most ontology modeling tools need to be downloaded and installed first, which increases the entry barriers for using the software. We identified requirements for an approach that is immediately available, easy-to-use, and independent of the device and interaction context. The most distinctive characteristic of our approach is the device-independent mode of operation for visual modeling of ontologies. Different input and output modalities are considered and brought into synergy for different devices and display sizes. The results of a preliminary user study indicate the benefits of the presented approach.

Since ontologies are multilingual conceptualizations of domain knowledge, i.e., different language tags are supported for literals to facilitate human-readable labels in different languages, a device-independent approach should also be independent of the language used in the modeling process. In our current implementation, we are limited to English.

The capacity of processing resources and the display size of touch devices like smartphones and tablets pose an additional limitation. With the growing size and complexity of ontology graph visualizations, their represented information tends to become hard to comprehend due to visual clutter and information overload. Due to the increased number of elements, this presents a higher processing load for interactive visualization, i.e., the transformations and animations of the graph. For small display sizes, the occlusion of rendering elements increases visual clutter. The semantic zooming approach for ontology graph visualizations [70] addresses these aspects and should also be considered for device-independent visual modeling.

The preliminary study participants also suggested adding a hierarchical view of the classes in the modeled ontology. In particular, this suggestion should be considered in future work because the user study revealed that participants without prior knowledge could out-perform those with experience in visual modeling and ontologies. Thus, a hybrid solution can reduce the entry barrier for domain experts, providing visual modeling in different graphical representations (i.e., hierarchical trees or node-link diagrams). However, providing more different modeling variants could also over-complicate the approach's usage and increase the entry barriers. Furthermore, synchronization mechanisms are then required to align the hierarchical view and the node-link diagram.

The proposed approach investigated the device-independent modes of operation based on touch and point and click interactions. Different modes such as speech and gesture recognition are also interesting further research topics. In the evaluation, we observed that all participants drew sketches similar to a node-link diagram using pen and paper. In summary, we hope that the approach will make the development of ontologies more attractive and foster engagement, particularly also in industrial settings, and additionally be useful to other researchers, ontology engineers, and domain experts.

4.2 A Comparative User Evaluation on Ontology Modeling Using Node-Link Diagrams

The emergence of several ontology modeling tools is motivated by the growing attention ontologies receive in scientific and industrial contexts. Thus, modeling of ontologies is often done collaboratively in joint efforts of knowledge engineers and domain experts. The available tools implement different ontology modeling paradigms such as text-based editors, graphical user interfaces with hierarchical trees and form widgets, and visual modeling approaches based on node-link diagrams. The different modeling paradigms range from direct text input, UML-based graphs [85], widget and hierarchical based GUIs [107], node-link diagrams [83, 103], to even hybrid solutions like TurtleEditor [88].

This section presents an empirical user study comparing a visual ontology modeling approach, based on node-link diagrams, with a modeling paradigm that uses hierarchical trees and form widgets. In particular, the user study compares the two ontology modeling tools: Protégé and WebVOWL Editor, each implementing one of the modeling paradigms. The involved participants were given ontology modeling tasks and answered reflective questions for the individual tools. The study involved 12 participants, including master students, Ph.D. students, and postdocs with computer science backgrounds. We evaluate the visual ontology modeling paradigm for node-link diagrams using WebVOWL Editor[8]. WebVOWL Editor exploits the VOWL notation, which is a well-defined visual notation for OWL ontologies. It is designed for the user-oriented representation of ontologies that is easy to understand [82]. This visual ontology modeling tool allows us to conduct our evaluation on different target groups, including non-expert users. The current implementation of WebVOWL Editor does not yet support all OWL constructs. However, it covers all required ones for our comparative evaluation.

The study indicates that visual ontology modeling, based on node-link diagrams, is comparatively easy to learn and is recommended especially for users with little experience with ontology modeling and its formalizations. We observed no apparent performance differences between the two modeling paradigms for more experienced users. Both approaches seem to have their pros and cons depending on the ontology type and the modeling context.

The diversity of different ontology modeling paradigms and tools also increased the interest in their benefits and drawbacks. Several evaluations investigating users' understanding of ontology representations and the effectiveness of different tools have been conducted. García et al. [108] conducted an evaluation on visual modeling, investigating the effectiveness and usability of the tool OWL-VisMod. The work of Katifori et al. [109] conducted a comparative user study of four ontology visualization tools. Users had to perform information retrieval tasks (e.g., finding the value of some property) with each tool. Completion time and post-interviews determine the effectiveness of each tool. According to that, Protégé Entity Browser is the most effective, then Jambalaya, TGViz, and OntoViz is the least effective. Fu et al. [110] compared the representation of ontologies with indented lists and node-link diagrams. Participants were asked to evaluate and create new mappings between ontologies using the two modeling paradigms. In this work, Fu et al. report that indented lists are more suitable for evaluating the mappings. In contrast,

[8]Tool and GitHub repository can be found at `https://w3id.org/webvowl/editor`

node-link diagrams are better suited for creating new mappings and showing an overview of the ontology. In a follow-up study, Fu et al. [111] used eye-tracking technology to investigate the differences between indented lists and graphs in more detail.

However, most of the existing evaluations focus on information retrieval tasks and on investigating how the comprised information of an ontology is communicated to the users. In contrast to comparing different representations of ontologies, we aim to fill the research gap by investigating the potential benefits and drawbacks of varying modeling paradigms for ontology creation. A pretest defines concept spaces that are used as modeling tasks in our evaluation. Participants had to model small ontologies using two ontology modeling tools: Protégé and WebVOWL Editor. Modeling completion times were measured, and additional questionnaires were used to determine the potential benefits and drawbacks of the individual tools.

4.2.1 Pretest

In advance of the user evaluation, we conducted a pretest. It includes **i)** the definition of concept spaces and **ii)** the identification of their individual difficulty levels, respectively. The pretest results are used to design a comparative user evaluation for visual ontology modeling using node-link diagrams and a hierarchical tree-based approach.

Concept Spaces for the User Study

Prerequisite to the pretest, we introduce five small concept spaces. These are defined by having a small generalized set of domain knowledge to evaluate different ontology modeling tools. The concept spaces define common, everyday knowledge, whereas each includes thirteen concepts. These concepts are associated with classes, subclasses, object properties, or datatype properties. Our defined set of domain knowledge consists of the following concept spaces: University, Zoo, Media, Family Tree, and City Traffic. The cognitive complexity of all concept spaces is balanced by:

1. Asking a person to define the concept spaces equal in hierarchical and graphical representations while created using any ontology modeling tool or even realized on paper. Repetitive iterations were performed on paper, defining the concepts for each domain knowledge.
2. Evaluating the difficulty levels for our defined set of domain knowledge measured by the time required for modeling a concept space.

University Space	Zoo Space	City Traffic Space	Media Space	Family Space
Staff Member	Animal	Vehicle	Media Network	Child
Person	Herbivores	Bus	News Channel	childs birthplace
Professor	Snow Leopard	model name	Daily News Channel	Family Tree
University	eats	Manufacturer	channel name	Female
Student	Carnivores	City Traffic	Sports Channel	Mother
Graduate Student	Cheetah	Public Vehicle	shows	Grandmother
has name	Zoo	Car	Local Sports Channel	Male
teaches	Goat	manufactured by	News Program	Gives Birth
Course	Grass	Private Vehicle	Channel	Father
Undergraduate Student	grass type	Train	International Sports Channel	Person
course name	has legs	manufacturing date	Airing Time	person name
has*	has*	has*	has*	has*
is a*	is a*	is a*	is a*	is a*

Figure 4.4: Classes and properties defined for each concept space, respectively.

The five concept spaces that were defined for the pretest and used in the study are shown in Figure 4.4. The concepts indicated with * could be used zero or multiple times.

Evaluating the Cognitive Complexity of the Concept Spaces

We evaluated the difficulty level for each of the defined concept spaces by recording the time required to perform the modeling task with Protégé. In total, four male participants without any visual, physical, or color blind impairment were involved in this evaluation. The participation in the pretest was voluntary, and the user's age was restricted to the range of 33 ± 6 years. This restriction assures that human motor performance is not affecting the modeling completion time. All participants had a profound experience with ontology modeling tools as they were affiliated with the field of Semantic Web, working as scientific researchers employed at Fraunhofer IAIS.

Method: The participants had to model the individual concept spaces which were presented to them in a tabular format (cf. Section 4.2.1 – Concept Spaces and Figure 4.4). The University Space was used as a training example. Therefore it was created by all participants in their first modeling task. The purpose of the training example was to make them comfortable using Protégé and allow them to configure the tool to their needs. The remaining modeling tasks were performed in alternating order, as shown in Table 4.5. This alternation was applied to avoid carry-over effects during the modeling tasks over time. The completion time for each modeling task was recorded in seconds and rounded off to the next smaller integer. All participants performed the experiment using a standard English (US) keyboard layout and an external mouse. The screen size was also kept the same to 16"9 inches with a 1920 × 1080 pixels resolution.

Results: The completion times for the individual concept spaces are presented in Table 4.5. These results indicate that the modeling tasks of the four concept spaces have, on average, a similar completion time. The qualitative findings from the pretest are:

1. During the modeling, two participants have crossed out the concepts in the table.
2. In general, we have noticed that the participants modeled classes, subclass hierarchies, and datatype properties in a similar fashion.
3. The participants varied in the way they have modeled object properties.

Table 4.5: Modeling completion times and the varying order of concept spaces.

Participant	Modeling Completion Times				Order of Concept Spaces
	Family Tree	**City Traffic**	**Media**	**Zoo**	
A	237	302	349	362	Zoo, Traffic, Media, Family
B	330	428	429	403	City, Zoo, Family, Media
C	389	183	361	270	Family, Media, City, Zoo
D	343	416	503	332	Family, City, Media, Zoo
Sum	1367	1329	1642	1299	
Mean	341.75	332.25	410.50	324.75	

4.2.2 Experimental Design

The evaluation design is based on the results we obtained from the pretest. We selected two concept spaces with the lowest mean difference between each other (i.e., Family Tree Space and City Traffic Space having a difference of 7.5 seconds in modeling times). In order to perform an empirical, comparative user study over visual modeling paradigm and hierarchical trees, participants were presented with the following nine tasks T1–T9:

T1: The participants had to fill out a demographic questionnaire, stating their *name, age, profession, experience in ontology modeling, experience with Protégé and WebVOWL, and any visual, physical, or color blind impairment.*

T2: Using Protégé, the participants had to model an ontology for the Family Tree Space or the City Traffic Space.

T3: Based on the modeled concept space in task T2, the participants had to fill out an After-Scenario Questionnaire (ASQ)[9] as a post-task.

[9]http://garyperlman.com/quest/quest.cgi?form=ASQ

T4: As a cued recall process [112], the participants had to highlight the concepts in a 6×4 table, which they thought they modeled using Protégé.

T5: Based on the modeled concept space in task T2, the participants had to fill out a Computer System Usability Questionnaire (CSUQ)[10] as a post-study task

T6: Using WebVOWL Editor, the participants had to model an ontology for the Family Tree Space or the City Traffic Space.

T7: Based on the modeled concept space in task T6, the participants had to fill out an ASQ questionnaire as a post-task.

T8: As a cued recall process, the participants had to highlight the concepts in a 6×4 table, which they thought they modeled using WebVOWL Editor.

T9: Based on the modeled concept space in task T6, the participants had to fill out a CSUQ questionnaire as a post-task.

Participants

Based on the demographic questionnaire, the 12 voluntary male participants were divided into two groups of experienced and non-experienced users (P_{G_1} and P_{G_2}). The first group P_{G_1} contained six participants with experience in ontology modeling. The second group P_{G_2} contained six participants without ontology modeling experience. The age of the participants was in the range of 25–36 years. Additionally, we restrict participants' age to the range of 31 ± 6 years to ensure that the human motor performance does not vary too much across the users. None of the participants had any visual or physical impairment. However, one participant mentioned that he was color blind. The participants included employees and students of Fraunhofer IAIS, University of Bonn, and RWTH Aachen with background in computer science.

Setup

To provide a homogeneous evaluation setup, all experiments were performed on a Dell Precision 3520 laptop with a standard English (US) keyboard layout and a screen size of 16"9 inches having a resolution of 1920×1080 pixels. An external mouse was used for navigation. The experiments were performed using Protégé (5.2.0) running on Ubuntu 18.04 and WebVOWL Editor using Mozilla Firefox or Google Chrome web browser. The study was conducted in the daily working environment of the participants.

[10]http://garyperlman.com/quest/quest.cgi?form=CSUQ

Procedure

The experiments were always supervised by the same person and performed using the setup provided by the conductor of the evaluation. All participants were given approximately ten minutes of training on both tools. The training sessions used *Media Space* and *Zoo Space*. These were selected due to significantly larger mean differences, meaning different difficulty levels. The remaining two concept spaces *Family Tree Space* and *City Traffic Space* were used in the actual ontology modeling tasks of the user study. Our pretest results indicate that these had significantly closer mean differences, meaning approximately the same difficulty levels.

All participants started the user study by answering the demographic questionnaire. We categorize the remaining eight tasks into two groups, T_{G_1} and T_{G_2}. Tasks T2–T5 are related to Protégé (T_{G_1}), whereas the tasks T6–T9 refer to WebVOWL Editor (T_{G_2}). After finishing the demographic questionnaire, each participant was asked to perform the tasks corresponding to one group first and then continue with the other group. The completion time for the modeling task was recorded in seconds and rounded off to the next smaller integer. As post-study questionnaires, we have chosen ASQ and CSUQ because of their high global reliability degree [113]. The ASQ measures *ease of task completion, satisfaction with completion time*, and *support of information*. The CSUQ contains 19 questions measuring *effectiveness, efficiency*, and *satisfaction* based on the ISO-9421-11 criteria [113]. Additionally, it measures *discriminability* based on the ISO/WD-9421-112 criteria [113]. *Guidance, workload*, and *error management* are measured w.r.t the Scapin and Bastien criteria [114]. Both questionnaires were answered using a Likert scale of 1 to 7, where 1 refers to strong disagreement, and 7 refers to strong agreement.

The 12 participants were divided into two groups (U_1 and U_2). The first group contained three experienced and four non-experienced participants. The second group contained two non-experienced and three experienced participants. This in-balanced assignment is a result of two invalid participations in the user group (U_2). However, the concept spaces are still counterbalanced, as illustrated in Table 4.6. The first user group (U_1) was asked to perform the Protégé specific tasks (T_{G_1}) first and then continue with WebVOWL Editor specific tasks (T_{G_2}). The second user group (U_2) performed the group tasks in inverse order. The alternating task order was done to avoid increasing or decreasing performance measurements over time. Table 4.6 shows the exact order of tool-specific tasks and the order of the concept spaces. The duration of the experiments for each participant was approximately 45–60 minutes.

Table 4.6: Order of tools and concept spaces presented to the participants.

Participant	Tool A	Concept	Tool B	Concept
1	Protégé	Family Tree	WebVOWL Editor	City Traffic
2	Protégé	City Traffic	WebVOWL Editor	Family Tree
3	Protégé	City Traffic	WebVOWL Editor	Family Tree
4	WebVOWL Editor	Family Tree	Protégé	City Traffic
5	WebVOWL Editor	City Traffic	Protégé	Family Tree
6	Protégé	Family Tree	WebVOWL Editor	City Traffic
7	Protégé	Family Tree	WebVOWL Editor	City Traffic
8	WebVOWL Editor	City Traffic	Protégé	Family Tree
9	WebVOWL Editor	City Traffic	Protégé	Family Tree
10	Protégé	Family Tree	WebVOWL Editor	City Traffic
11	WebVOWL Editor	Family Tree	Protégé	City Traffic
12	Protégé	City Traffic	WebVOWL Editor	Family Tree

4.2.3 Results and Discussion

The results of our user study comprise of 1) performance scores for the ontology modeling tasks, 2) scores for the participants recalling the concepts of the modeled ontology, and 3) the scores for user satisfaction for the two ontology modeling tools. The study manual, the definition of concept spaces, and the evaluation data are available on GitHub[11].

Performance Scores for Ontology Modeling

The performance scores for tasks T2 and T6 were calculated based on the required time to model an ontology (cf. Section 4.2.2). The completion times are illustrated in Table 4.7 and indicate that WebVOWL Editor performed better in comparison to Protégé. On average, all 12 participants completed the ontology modeling task 18.7 seconds faster using WebVOWL Editor. The experienced (P_{G_1}) and non-experienced (P_{G_2}) participants performed respectively 26.2 and 11.4 seconds faster using WebVOWL Editor. The average difference between the completion times for the individual tools $M_{avg} = \frac{1}{12}\sum_{i=1}^{12} T2(i) - T6(i)$ was 23.4 seconds. Where $T2(i)$ and $T6(i)$ denote the time participant i required to model an ontology for the individual task, respectively. The standard deviation of differences was 80.54 seconds.

Figure 4.5 a) indicates that Protégé had less variance, whereas WebVOWL Editor had more variance in results. For two participants, the experiment was repeated as their

[11] https://github.com/vitalis-wiens/VisualOntologyModelingEvaluationData

Table 4.7: Average time required to model an ontology for both tools.

	Mean Scores		Standard Deviation	
Participant Type	**Protégé**	**WebVOWL Editor**	**Protégé**	**WebVOWL Editor**
All Participants (12)	386.5	367.8	89.84	149.06
Experienced (6)	364.5	338.3	105.00	136.76
Non-Experienced (6)	408.5	397.1	74.63	167.64

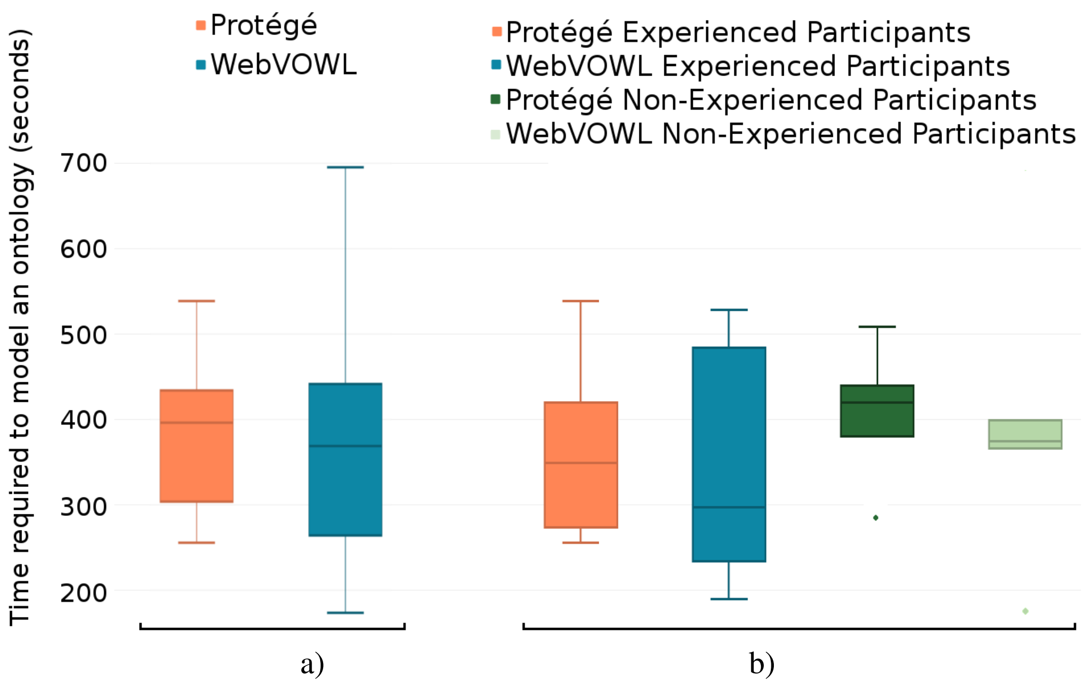


Figure 4.5: Required modeling time as a box plot diagram. a) Modeling time for all participants. b) Modeling time for different participant groups based on experience.

modeling task was interrupted. Thus the outliers that are shown in Figure 4.5 b) can result from the experiment repetition. Additionally, Figure 4.5 b) indicates that experienced participants had a higher variance with a broader spread of the central box while using WebVOWL Editor and Protégé, that is 250 and 146 seconds. In contrast, for non-experienced participants, the spread of the central box for WebVOWL Editor and Protégé is 60 and 33 seconds, respectively. Therefore, we can infer that the broader spread for experienced participants resulted from their diversified experience using the tools. For the non-experienced participants, a much lower spread denotes that the performance of participants was similar. While executing the modeling tasks, with WebVOWL Editor, a lower central box for the non-experienced participants reveals that users without prior experience tend to perform better using WebVOWL Editor than Protégé.

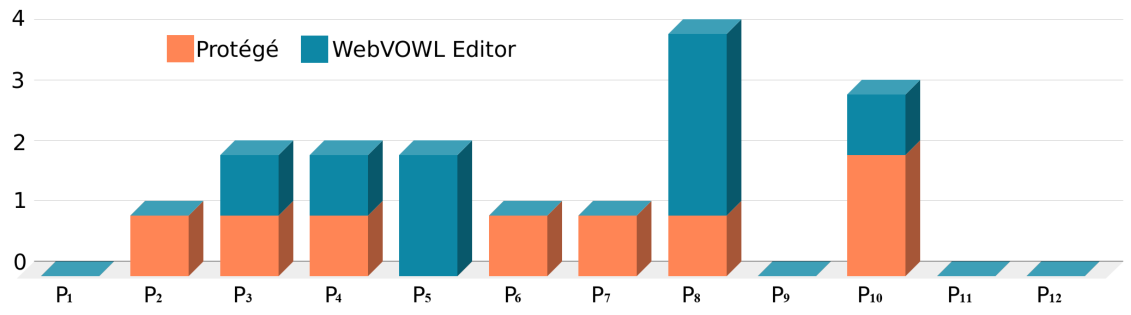


Figure 4.6: Incorrectly highlighted concepts per participant (P_i) for the two tools.

Cued Recall Scores

The cued recall scores were measured by the number of correctly highlighted concepts for tasks T4 and T8 (cf. Section 4.2.2). While measuring the correctness, *is a* and *has* concepts were not considered. These were allowed to be used repetitively or not at all.

Figure 4.6 shows the number of incorrectly highlighted concepts for individual participants. In total, the number of incorrectly highlighted concepts was eight for each tool. With respect to highlighting concepts, seven participants were incorrect for task T4, whereas five were incorrect for task T8. These results indicate that fewer participants were incorrect with WebVOWL Editor than with Protégé.

User Satisfaction Scores

ASQ — Figure 4.7a) indicates that the participants were more satisfied with the *ease of completing the task* and the *time it takes to complete a task* when using WebVOWL Editor. The participants were equally satisfied in using the two tools for the *support information provided by the tool*. Figure 4.7b) indicates that the experienced group (P_{G_1}) had a higher score for *ease of completing the task* and *time it takes to complete a task* using WebVOWL Editor. However, these results also indicate that the *support information provided by the tool* for WebVOWL Editor requires improvement. The results for the non-experienced group (P_{G_2}) show that WebVOWL Editor was perceived as requiring less *time to complete a task*, and it provided better *support information*.

CSUQ — WebVOWL Editor scored better in 16 of 19 CSUQ questions. Protégé scored better in questions related to *the number of system capabilities, the information provided by the system*, and *if they can effectively complete their work using the system*. Protégé scored 5.4, 3.75, and 5.9, whereas WebVOWL Editor scored 4.9, 3.5, and 5.75, respectively. Based on the different participant groups (P_{G_1} and P_{G_2}), the scores show that P_{G_2} still

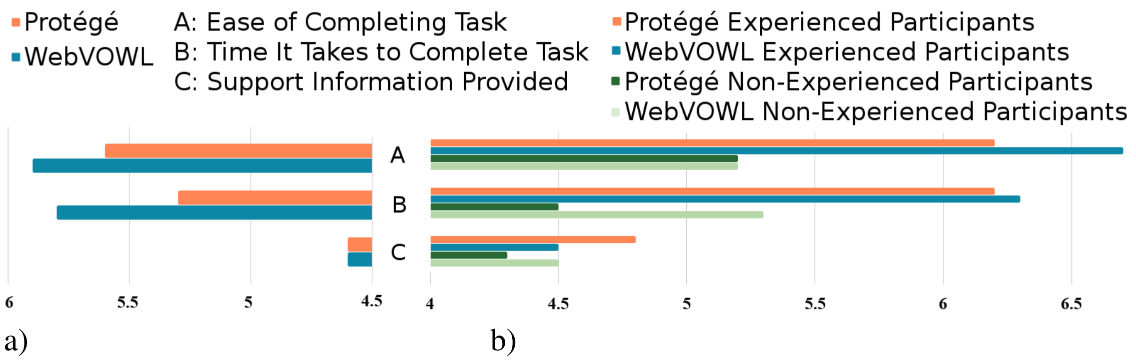


Figure 4.7: ASQ: a) Scores for all participants. b) Scores for different participant groups.

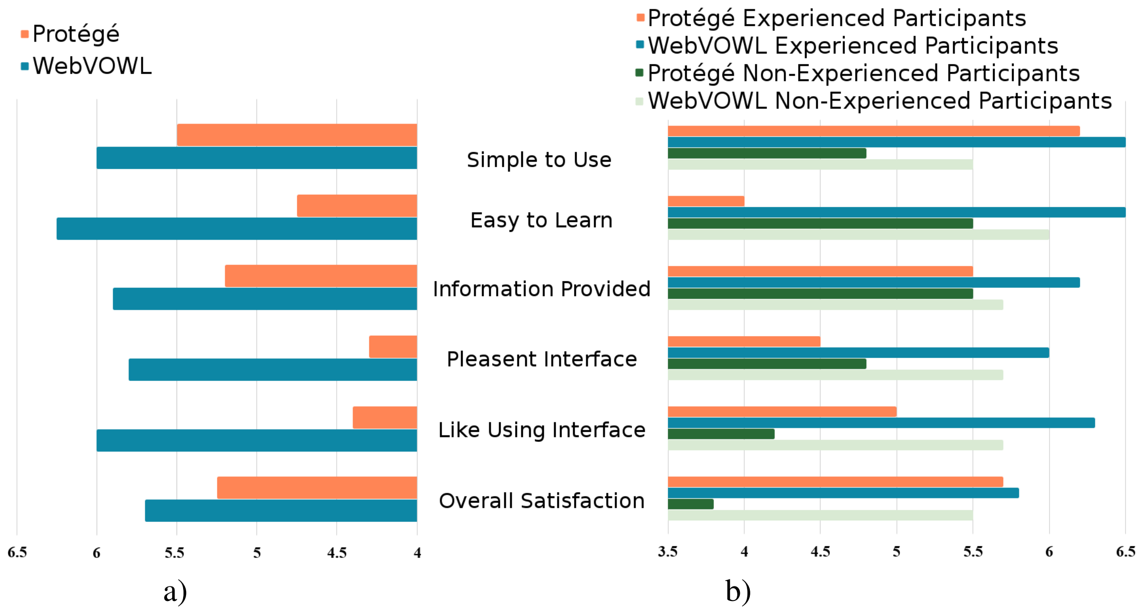


Figure 4.8: CSUQ: a) Scores for all participants. b) Scores for different participant groups.

rated WebVOWL Editor better for *effectively completing their work* and the *number of system capabilities* with a score of 5.3 and 5.5, whereas Protégé scored with 5.2 and 5.2. Six questions for which the results had a significant difference between the two tools are shown in Figure 4.8. The CSUQ results indicate that both participant groups P_{G_1} and P_{G_2} rated WebVOWL Editor better in terms of usability.

4.2.4 Summary of the User Study

This section compared a visual ontology modeling approach using WebVOWL Editor with a hierarchical tree, GUI-based modeling using Protégé. Visual ontology modeling approaches, particularly in the form of node-link diagrams, help non-expert users to get directly involved in ontology modeling without any prior experience. We introduced five small concept spaces (cf. Section 4.2.1) and determined their cognitive complexity using a pretest. The results of the pretest indicate similar difficulty levels for *City Traffic Space* and *Family Tree Space*. Thus, these two were used in the ontology modeling tasks. Participants had to perform ontology modeling tasks, reflective question answering tasks, and filled out additional ASQ and CSUQ post-task questionnaires.

The results of the experiment (cf. Section 4.2.3) indicate that overall the participants were more efficient, they had a better understanding of the model, and they were more satisfied using WebVOWL Editor than Protégé. The mean performance measures for both tools had a minor difference with WebVOWL Editor having a better performance. The performance was much better for the non-expert user group (P_{G_2}), highlighting a low learning curve with a good performance rate for novice users. The results were not significant for the expert user group (P_{G_1}) and had high variance due to their prior experience with both tools, as shown in Figure 4.5b). WebVOWL Editor scored better in the following usability areas: *ease of task completion, time taken for task completion, ease of learning the system, simplicity of using the system, pleasantness of the interface, likeability to the interface,* and *overall user satisfaction for the system.*

The VOWL notation is designed for a user-oriented representation of OWL ontologies for different user groups. WebVOWL Editor is designed for device-independent ontology modeling and thus realizes minimalistic user interactions, allowing it to be used on touch devices. Visual modeling paradigms that allow for better mental map preservation, the VOWL notation, and the minimalistic user interactions are beneficial for the performance of WebVOWL Editor. However, due to the small sample size, the results indicate only a minor increase in performance. Thus we suggest a follow-up study with an increased number of participants to at least twenty, as suggested by Nielsen [115]. It improves the confidence interval and reduces the margin of error. We also propose controlling the prior experience with modeling tools, thus, reducing the variance and improving the comparison of results between the tools.

4.3 Chapter Summary

In this chapter, we have presented a device-independent visual modeling approach for ontology modeling. The approach is designed to reduce entry barriers for domain experts and other user groups to get more directly involved with ontology modeling. Additionally, we have presented a list of requirements for a device-independent visual modeling approach that have been derived from sessions with domain experts in the context of the research project GRACeFUL and industry workshops conducted from the manufacturing and healthcare domains. We presented a preliminary user evaluation comparing the approach against two other web-based applications.

Visual modeling approaches in Semantic Web contexts have three main challenges: **i)** mapping the information contained in ontologies to graphical primitives facilitating the understanding of the comprised information, **ii)** provide guidance during the modeling process, and **iii)** map the visual representation back to an ontology. We address the challenges as follows: **i)** We use the VOWL notation for mappings of ontology information to visual primitives. **ii)** Build-in constraints guide the user during the modeling process. **iii)** Using the well-specified VOWL notation, we map the visual representation back to ontology elements enabling users to export the ontology as a file in the Turtle serialization.

A follow-up user evaluation compared visual modeling using node-link diagrams against hierarchical trees and form widgets. Most of the existing evaluations focus on information retrieval tasks and on investigating how the comprised information of an ontology is communicated to the users. In contrast to comparing different representations of ontologies, we investigated the potential benefits and drawbacks of varying modeling paradigms for ontology creation. The study indicates that visual ontology modeling, based on node-link diagrams, is comparatively easy to learn and is recommended especially for users with little experience with ontology modeling and its formalizations.

The evaluation results provide empirical evidence to answer **RQ1** that device-independent visual modeling approaches facilitate the creation and editing processes of Semantic-Web-Based knowledge structures. In particular, visual modeling approaches in the form of node-link diagrams allow for engaging users that are less familiar with ontology modeling and OWL formalizations. Furthermore, our visual modeling approach focuses on device independence, facilitating many interaction contexts, ranging from classical desktop settings to mobile scenarios in meetings, workshops, and on business trips. In the following two chapters, we investigate how to increase further the understanding of the information contained in Semantic-Web-Based knowledge structures using visual representations.

CHAPTER 5

Customizable Graph-Based Visual Representations of Ontologies

Visualizations support human cognition by exploiting human's ability to understand complex data through visual representations. While various visualization methods and tools exist, suitable visualizations are highly dependent on the use cases and the targeted user groups. Users often require customizations to reflect their previous experiences with other visualization methods and tools. We introduce a methodology for customizable visual representations of ontologies. The methodology defines visual representation models using the OWL annotation mechanisms. Annotation ontologies address different aspects of the visualization using the separation of concerns paradigm. The use of `owl:imports` statements links ontologies with visual definitions for their depiction. We showcase the applicability of the methodology by introducing GizMO, a representation model for graph-based ontology visualizations. Two applications show the utilization of GizMO and the variety of achievable visualizations. The applications are designed to reduce the textual crafting of annotation ontologies and showcase the interoperability of the methodology.

This chapter addresses the following research question:

Research Question 2 (RQ2)

How can we improve understanding of Semantic-Web-Based knowledge structures using interactive and user-centered visualizations?

The contributions of this chapter are the following:

- Methodology for a customizable representation model for ontologies.
- Graph visualization meta ontology (GizMO) for node-link diagram visualizations.

This chapter is based on the following publication: [116]

The remainder of this chapter is structured as follows: Section 5.1 introduces the motivation and requirements for customizable graph-based visualizations. Section 5.2 presents a methodology for a customizable representation model for visualizations of ontologies. Section 5.3 showcases the conceptual realization of the methodology for node-link diagrams in the form of a Graph Visualization Meta Ontology (GizMO). Section 5.4 presents two applications and demonstrates achievable visual representations using GizMO. Section 5.5 summarizes the approach with an outlook on the research question **RQ2**.

5.1 Motivation and Requirements

The development, exploration, communication, and sense-making of ontologies can be facilitated using visual representations [16]. However, a visual representation model needs to reflect the user's mental model to provide a suitable visualization. The challenge with most approaches is grounded in their design. Visualization methods and tools are typically created for a particular use case with a specific definition for the representation model. Ideally, this visual representation model corresponds to the user's mental model and facilitates understanding the underlying information. With the increasing attention of the Semantic Web in academic and industrial contexts, various user groups are involved in the ontology modeling process. Thus, a single visual representation model will diverge from expectations and previous experiences with other visualization tools for some user groups. The number of visualization methods, tailor-suited tools, along with the requirements and necessity for customization indicate that a *one-size-fits-all* solution is challenging, if not impossible, nor feasible, to realize. In order to satisfy the varying demands of users and use cases, suitable visualizations require customizable representation models.

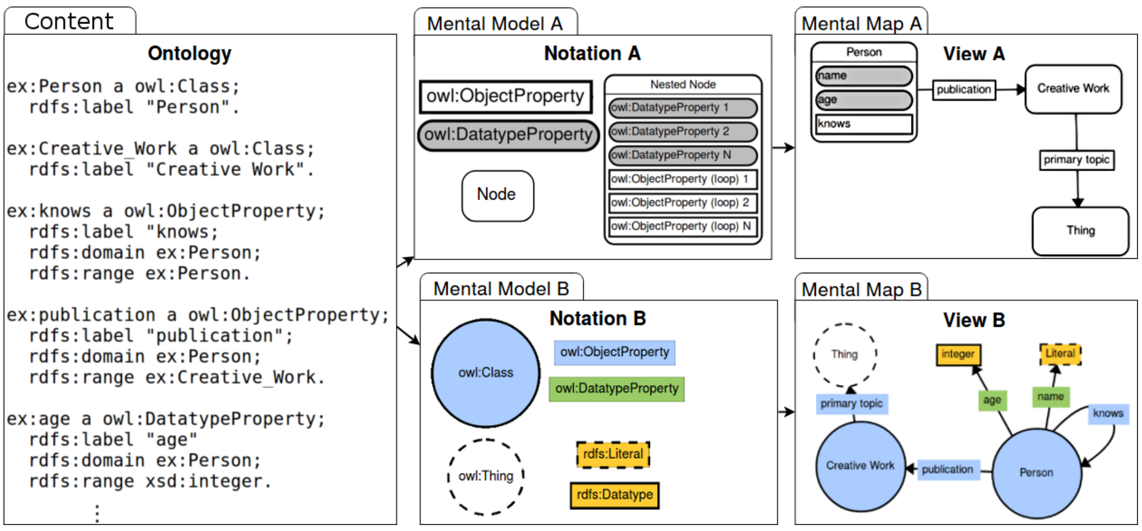
Without any loss of generality for the methodology, this section focuses on domain ontologies and their visual representations as node-link diagrams. As illustrated in Figure 5.1, the two main visual characteristics of different methods and tools are *visual appearance* and *spatial arrangement*, i.e., the graphical notation used to draw the ontology graph and the layout of nodes and edges. Thus, a customizable visual representation model needs to address the following requirements: **i)** provide the customizable visual appearance of rendering elements in order to coincide with the user's *mental model*; **ii)** provide spatial information and visibility status of rendering elements in order to coincide with the user's *mental map*; **iii)** provide the means to represent and share the definition of visualizations.

Visual Appearance – Visual notations formally define the visual appearance of elements for the depiction of ontology elements, such as `owl:Class`. Examples of visual notations for ontologies are VOWL [82] and Graffoo [83]. Also, UML is often used to represent ontologies [84]. Although UML has a standard visual notation, various styles exist, such as the visual representation of ontologies with TopBraid Composer [85], a UML version of the VOWL notation [86], or the UML mapping of the NeOn Toolkit [87].

Graph-based visualizations can be categorized into name-label-only and nested visualizations [17]: Name-label-only visualizations depict the elements as individual labeled nodes and links. Nested visualizations (e.g., UML) aggregate information and visualize a list of attributes inside the corresponding node. Thus, to coincide with the user's *mental model*, a representation model has to provide a customizable notation that defines visual characteristics and also encodes the depiction of aggregated elements.

Spatial Information – Spatial information is essentially necessary to preserve the *mental map* of users [18]. The spatial arrangement of elements in different layouts (e.g., hierarchical trees or circular layouts) can facilitate the organization of information and sense-making. The preservation of a user-defined layout is only obtainable when spatial information is attached to the domain ontology. The spatial information has to correspond to the used notation in order to reconstruct the visualization correctly. For example, nested visualizations typically do not encode spatial information for aggregated elements. Therefore, using spatial information of a nested visual notation in a different one can result in an invalid spatial assignment of elements. Consequently, a visual representation model has to provide spatial information that corresponds to the used notation and the domain ontology for which it has been created.

Glyph Specific Information – A glyph is a collection of rendering primitives. For example, a glyph could be a composition of a shape (e.g., circle) and a label text. Glyph specific information has to address additional properties, such as the visibility status of a glyph. With the growing size and complexity of an ontology, visualizations become harder to read due to visual clutter and information overload. Various ontology visualization tools address this by providing filtering mechanisms to reduce the information load for humans' limited cognitive capacity [117]. Thus, balancing the cognitive load requires strategies for determining parts of the ontology to be visualized or to be hidden.



```
ex:Person a owl:Class;
  rdfs:label "Person".

ex:Creative_Work a owl:Class;
  rdfs:label "Creative Work".

ex:knows a owl:ObjectProperty;
  rdfs:label "knows;
  rdfs:domain ex:Person;
  rdfs:range ex:Person.

ex:publication a owl:ObjectProperty;
  rdfs:label "publication";
  rdfs:domain ex:Person;
  rdfs:range ex:Creative_Work.

ex:age a owl:DatatypeProperty;
  rdfs:label "age"
  rdfs:domain ex:Person;
  rdfs:range xsd:integer.
        ⋮
```

Figure 5.1: Separation of concerns into notations and views. Notations define how OWL constructs are depicted. Views provide spatial position assignment and visibility status.

Suitable ontology visualizations may require customization for individual glyphs. An often requested feature is the modification of visual characteristics for some elements of the domain ontology. These could be modifications of shapes and colors for highlighting or grouping these elements in the visualization. Also, the replacement of selected shapes and labels with icons or images is sometimes requested. The individual glyph modifications address the aspect of *preattentive processing* capabilities of humans in order to facilitate target detection tasks, providing an effortless information perception and organization. Thus, a representation model needs to provide customizable visual definitions for elements of the domain ontology.

Representation, Governance, and Management – Some of the available tools, such as Cytoscape [94], already partially fulfill the described requirements. However, the proposed solutions are typically restricted to a specific tool and visualization method. A methodology that defines visual representations in the form of OWL annotation ontologies can overcome such boundaries. Nevertheless, such a methodology has to ensure the following criteria:

1) **Reusability:** The methodology should provide the means to recreate and modify existing notations or even design new ones so that these can be shared and reused. The methodology should provide the means to define spatial positions for elements of the domain ontology. Additionally, it should provide the means to customize glyph specific information (e.g., visibility status, shape, and color).

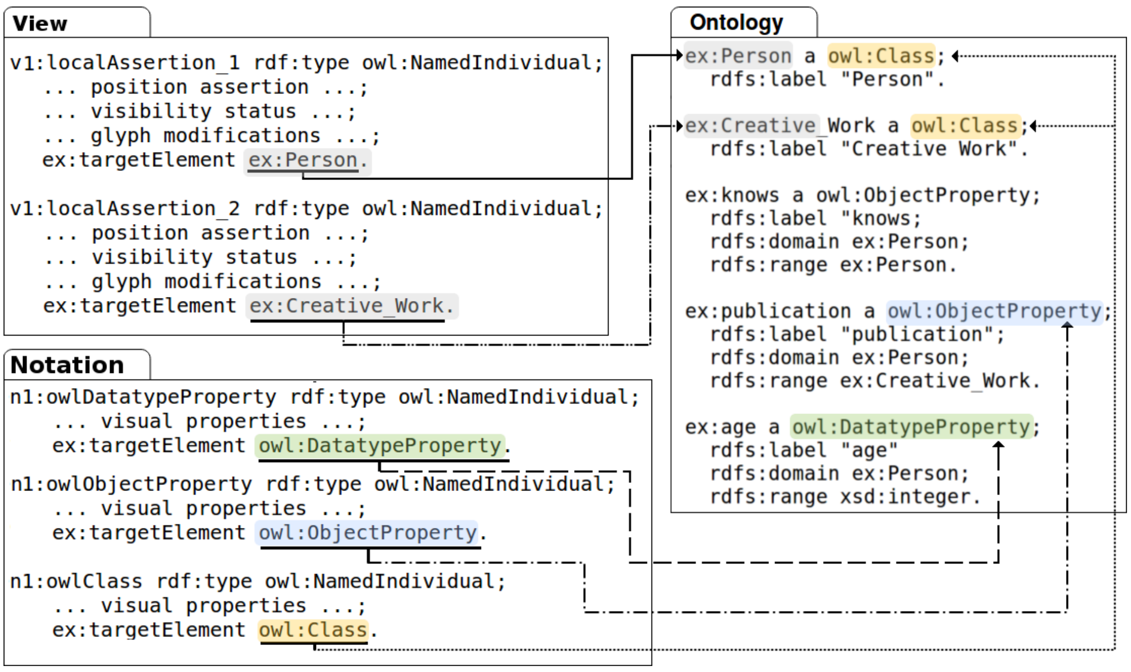
2) **Preserving the originality of the domain ontology:** Our methodology uses annotations to enrich elements with definitions for their visual depiction. However, directly attaching visual descriptions to the domain ontology or its elements results in an unwanted solution: the modification of OWL constructs and elements of the domain ontology. Thus, the methodology should provide the means to bind meta information to elements of the domain ontology without the need to modify them directly.
3) **Separation of concerns:** The separation of concerns plays an essential role in fostering flexible visual representations. The methodology should provide the means for the flexible exchange of specific visual properties. Our methodology addresses different aspects of the visualization on two information layers that are represented as annotation ontologies. These annotation ontologies enrich domain ontologies with visual definitions for their depiction. Domain ontologies and their visual definitions are combined using `owl:imports` statements.

5.2 Methodology

In Semantic Web contexts, numerous visual representation models have been developed. However, these are typically restricted to the corresponding visualization method and tool. We introduce a methodology that can overcome method and tool-specific boundaries and enables customizations for ontology visualizations. Furthermore, we provide a more detailed discussion for individual parts of the methodology.

Ontologies are not designed with the focus on presenting information to humans [4]. They are created and shared as file representations in various serializations (e.g., Turtle, N3, etc.), supporting the semantics-aware exchange and processing of information. Visualization tools *parse* the textual definition and depict the ontology accordingly to a specific visualization method. Thus, *any* ontology visualization tool has a parsing mechanism and is capable of interpreting ontologies.

Our methodology exploits the fact mentioned above and defines visual representations as annotation *ontologies*. These annotation ontologies can be used with arbitrary domain ontologies using `owl:imports` statements. Thus, a separation of concerns is realized for the visual abstraction layer. The visual abstraction is divided into two information layers. The first layer provides *global* visual descriptions for OWL constructs. Accordingly, this layer is independent of the visualized domain ontology. Conceptual elements will be depicted based on their type (`rdf:type`) assertion to OWL constructs.



```
v1:localAssertion_1 rdf:type owl:NamedIndividual;
   ... position assertion ...;
   ... visibility status ...;
   ... glyph modifications ...;
   ex:targetElement ex:Person.

v1:localAssertion_2 rdf:type owl:NamedIndividual;
   ... position assertion ...;
   ... visibility status ...;
   ... glyph modifications ...;
   ex:targetElement ex:Creative_Work.
```

```
n1:owlDatatypeProperty rdf:type owl:NamedIndividual;
    ... visual properties ...;
    ex:targetElement owl:DatatypeProperty.
n1:owlObjectProperty rdf:type owl:NamedIndividual;
    ... visual properties ...;
    ex:targetElement owl:ObjectProperty.

n1:owlClass rdf:type owl:NamedIndividual;
    ... visual properties ...;
    ex:targetElement owl:Class.
```

```
ex:Person a owl:Class;
  rdfs:label "Person".

ex:Creative_Work a owl:Class;
  rdfs:label "Creative Work".

ex:knows a owl:ObjectProperty;
  rdfs:label "knows;
  rdfs:domain ex:Person;
  rdfs:range ex:Person.

ex:publication a owl:ObjectProperty;
  rdfs:label "publication";
  rdfs:domain ex:Person;
  rdfs:range ex:Creative_Work.

ex:age a owl:DatatypeProperty;
  rdfs:label "age"
  rdfs:domain ex:Person;
  rdfs:range xsd:integer.
```

Figure 5.2: Illustration of the target property linking conceptualization.

The second layer provides *local* visual descriptions for elements from the visualized ontology. This layer is designed to describe additional information, such as spatial position and visibility status. Accordingly, this layer is bound to a particular ontology, and its visual descriptions are only valid *locally* for the individual conceptual elements of the visualized ontology. In this chapter, the terms *mental model*, *global layer*, and *notation* refer to the definition of visual properties for OWL constructs. The terms *mental map*, *local layer*, and *view* correspond to visual properties for conceptual elements of the visualized ontology.

Inspired by the Web Annotation Data Model [95], the methodology uses targeting properties to link representational definitions with OWL constructs and individual elements from the visualized ontology. Visual properties of distinct elements are organized in instances of type `owl:NamedIndividual`. All properties used in the methodology are of type `owl:AnnotationProperty`. As shown in Figure 5.2, these instances link visual descriptions to corresponding elements. Thus, this approach preserves the originality of OWL constructs and the elements of the domain ontology.

Notations – Notations describe the visual depiction of OWL constructs on the global layer. Named individuals provide the grouping of visual properties linked to the corresponding OWL constructs using the targeting properties. Since visual notations are defined as annotation ontologies, they can be easily exchanged by adjusting the import statements.

While this work focuses on *domain ontologies* and their graph-based representations, the methodology itself is not restricted to only those types of visualization methods. Various visualization methods, such as treemaps and Euler diagrams, could be defined when adequate notations are created and visualization frameworks provide support for these types of visualization methods. Furthermore, conceptualization from upper-level ontologies can be enriched with visual descriptions in the same fashion, using the targeting link approach of the Web Annotation Data Model [95].

Views – Views are annotation ontologies that hold a set of named individuals targeting conceptual elements of the visualized ontology. Views provide additional information, such as the spatial position and visibility status of elements. However, the used notation is essential to obtain valid assertions for the position of the elements. In our methodology, we enforce views to provide the information for which notation they were created.

Furthermore, views provide *optional* modification of glyph specific information. A glyph is a collection of rendering primitives (e.g., shape and label text) for OWL constructs. Since views encode information about the used notations, modified glyphs are created by applying the visual description of the corresponding OWL construct and then overwriting visual properties in correspondence with the provided glyph modification. For example, elements from the domain ontology can be visually highlighted and grouped by adjustments of their visual attributes (e.g., colors, shapes, and sizes).

Containers – The design of the methodology enables the orchestration of notations and views. Domain ontologies can be enriched with metadata using `owl:imports` statements (if the annotation ontologies are exposed under a dereferenceable URI). Annotation ontologies for visual representations can be exchanged, shared, reused, and adjusted for users' current needs. Within the combined model, the disambiguation between conceptual and visual elements can be realized through different namespaces. The methodology's conceptualization encodes visual attributes in instances solely of type `owl:NamedIndividual` and annotation properties.

5.2.1 Methodology Discussion

The methodology is an abstract conceptualization for the description of customizable visual representations of ontologies. The central aspect of the methodology is its utilization of OWL for definitions of visual representation models. Furthermore, it provides conceptualizations for separation of concern in terms of annotation ontologies.

The organization of visual properties in `owl:NamedIndividuals` and their use of linking properties provide the means to enrich any resource with a description for its visualization. Thus, the full spectrum of resources, such as OWL constructs and conceptual elements from ontologies, can be addressed.

The methodology provides no restrictions concerning which annotation properties are grouped and instantiated in named individuals. This aspect is governed by an implementation of the methodology, such as GizMO. The expressivity is not restricted by the methodology but by the implementation and the corresponding visualization framework.

The aspect of usability should be governed by such applications and solely be their responsibility. We argue that the corresponding frameworks should address the variety in technological stacks. Visualization frameworks have to interpret the representation models and create the depictions in their specific programming languages. Since the methodology has no restrictions, it could also describe user interactions, such as visual modifications on mouse hovering or even descriptions for mouse click events. Nonetheless, the methodology is only as viable as its realization and implementation in visualization frameworks.

The methodology's constraints are its use of annotation properties for the assignment of visual property values and `owl:NamedIndividuals` for the grouped assignment. The identification of `rdfs:domain` restrictions can realize the disambiguation between conceptual and visual annotation properties. Conceptual instances are of type `owl:NamedIndividual`. However, these are additionally assigned to other OWL constructs. Thus, due to multiple type assertions, these instances belong conceptually to the element defined in the ontology. Listing 5.1 shows multiple type assertions for the disambiguation between different instance types.

```
ex:Author_1   a  owl:NamedIndividual, foaf:Person .
ex:Example_1  a  owl:NamedIndividual .
```

Listing 5.1: Disambiguation between instances for visual definitions and instances of the domain ontology.

Author_1 becomes an instance of the class `Person`. In contrast, Example_1 is an instance that belongs to `owl:Thing`. We argue that instances of `owl:Thing` and their asserted annotation properties provide a suitable disambiguation mechanism. Such abstract information does not provide value for the conceptualization of ontologies. Thus, the methodology offers full coverage and disambiguation conceptualizations. Usability and expressivity are aspects that are addressed by corresponding implementations.

5.3 GizMO

The graph visualization meta ontology (GizMO) is a representation model for the definition of customizable graph-based ontology visualizations. Based on the methodology, GizMO defines visual representations as OWL *annotation* ontologies. This section provides an overview of design decisions, technical realization, and discusses coverage and limitations.

5.3.1 Preliminaries

GizMO builds on the following concepts:
OWL Constructs – The language constructs of RDF(S) and OWL
(e.g., `owl:Class`, `owl:objectProperty`, `rdfs:subClassOf`).
GizMO Core Ontology – Set of defined annotation properties, along with value restrictions. Besides visual properties (e.g., shape, color, and position), it defines further annotation properties used by the representation model.
Annotation Object – Based on the methodology, annotation properties are grouped in instances of type `owl:NamedIndividual`. These instances link visual definitions to corresponding elements. This approach ensures the separation between elements of the *domain* ontology and corresponding visualization definitions. Targeting properties prevent unnecessary manipulations of elements. These named individuals are extended with an additional annotation property that defines their annotation object type. The annotation object types enable the disambiguation of the named individuals for different parts of the representation model. The GizMO representation model uses five annotation object types:

- **Glyph Annotation Object:** Organizes the set of annotation properties that address the visual appearance of OWL constructs.
- **Visualization Annotation Object:** Organizes the set of annotation properties for the visual representation of conceptual elements from the domain ontology, including their spatial position, visibility status, and optional modified glyph information.
- **Triple Annotation Object:** Identifies triples from the domain ontology and asserts the visual annotations for the corresponding subject, predicate, and object element.
- **Notation Annotation Object:** Holds additional information for the notation, such as a canvas background color.
- **View Annotation Object:** Holds additional information for the view, such as the used notation and viewport configuration (e.g., zoom factor).

Notation – A notation defines the visual appearance of OWL constructs on the *global* layer. It provides a *notation annotation object* and a set of *glyph annotation objects* that provide the visual appearance definitions for different OWL constructs.
View – A view defines the spatial assertion of conceptual elements of the domain ontology on the *local* layer. It provides a *view annotation object*, and two different sets of annotation objects, i.e., *triple annotation objects*, and *visualization annotation objects*.

5.3.2 Visual Graph Mapping

Ontology visualization methods and tools apply a mapping that provides a formal definition for the visual representation of OWL constructs. Graph-based visualizations, such as node-link diagrams, represent the concepts of an ontology by a graph $G(N, L)$, where classes typically map to the set of nodes N and their interrelations are described by the set of links L using the terms of the ontology. These nodes and links have a visual component for the graphical depiction based on the mapping. However, even for two-dimensional graph-based visualizations, these mappings provide deviations for nested and non-nested representations in the final depiction of the ontology (cf. Figure 5.1, views A and B). Furthermore, OWL provides constructs that can be mapped differently, e.g., `a:ClassA owl:equivalentClass b:ClassB`. One representation could map `a:ClassA` and `b:ClassB` to nodes in the graph, having `owl:equivalentClass` as a link between the two nodes. Another representation could merge the two classes into one node. Thus the OWL construct `owl:equivalentClass` does not have a single visual component but instead has a mapping description for the involved elements.

Some visual notations provide for OWL constructs a visual component and a mapping description. For example, `rdfs:Literal` is a resource that maps to a node with a visual component. Thus, all datatype properties with this `rdfs:range` restriction will provide a link to this *single* node. Depending on the domain ontology and its complexity, such a *simple* mapping could result in an overcrowded and cluttered visualization. Correspondingly, node splitting or glyph multiplications are performed by some notations, e.g., VOWL performs glyph multiplications for literals, datatypes, and `owl:Thing`.

Ontology mappings can be categorized into three groups: **i)** visual-component-only creates for an OWL construct a node or a link; **ii)** mapping-description-only defines how axioms are handled (e.g., `owl:equivalentClass`); **iii)** combination of **i)** and **ii)**. The methodology does not employ definitions for the mapping of OWL constructs. This design decision transfers the responsibility of the mapping to the corresponding notation and

Table 5.1: OWL constructs and corresponding mappings currently supported by GizMO.

OWL Construct	Mapping
Nodes	
owl:Class	Visual-component-only
owl:Thing	Combination (glyph multiplication)
Links	
owl:ObjectProperty	Visual-component-only
owl:DatatypeProperty	Visual-component-only
rdfs:subClassOf	Combination (glyph multiplication)
Datatypes	
rdfs:Literal	Combination (glyph multiplication)
rdfs:Datatype	Combination (glyph multiplication)
Other	
rdfs:label	Combination (label in domain node)
rdfs:domain	Mapping-description-only
rdfs:range	Mapping-description-only

maintains the flexibility of the methodology for various notations. The current conceptualization of GizMO does not support such customizable mapping definitions. GizMO is limited to a subset of OWL constructs and a fixed implicit mapping. Table 5.1 provides an overview of support OWL constructs and their mappings.

While GizMO supports only a limited set of OWL constructs and corresponding mappings, its conceptualization provides five additional customizable visual elements that mitigate the current constraints. Default elements for nodes and properties describe a visual-component-only mapping for OWL constructs that are not supported by GizMO. Unsupported datatypes are represented by a corresponding default element that defines their visual component and enforces a glyph multiplication. Furthermore, GizMO provides descriptions for collapse/expand mechanisms of multiple links between two nodes. The visual representation of collapsed links is provided by one of the additional elements. The mappings for nested visualizations (e.g., UML) are implicitly defined in GizMO and are represented by a nested node description. The use of nested visual representations is defined in the *notation annotation object*.

GizMO has been designed to showcase the applicability of the methodology. Regardless of the set of supported OWL constructs and the implicit mappings, GizMO provides already customizable visualizations of ontologies that can be used to recreate existing notations, such as UML and VOWL.

5.3.3 Technical Realization and Design Decisions

The technical realization is conceived in correspondence with the methodology. An annotation ontology, the GizMO core ontology[1], defines annotation properties for the GizMO representation model. This ontology provides value restrictions and comments, addressing purpose and usage for each annotation property. Furthermore, it defines five annotation object types (i.e., glyph, visualization, triple, notation, and view annotation object) for the disambiguation of `owl:NamedIndividuals`. GizMO defines additionally rudimentary interaction descriptions for hovering on glyphs. A subset of annotation properties indicating their purpose is shown in Table 5.2.

Table 5.2: Subset of GizMO annotation properties.

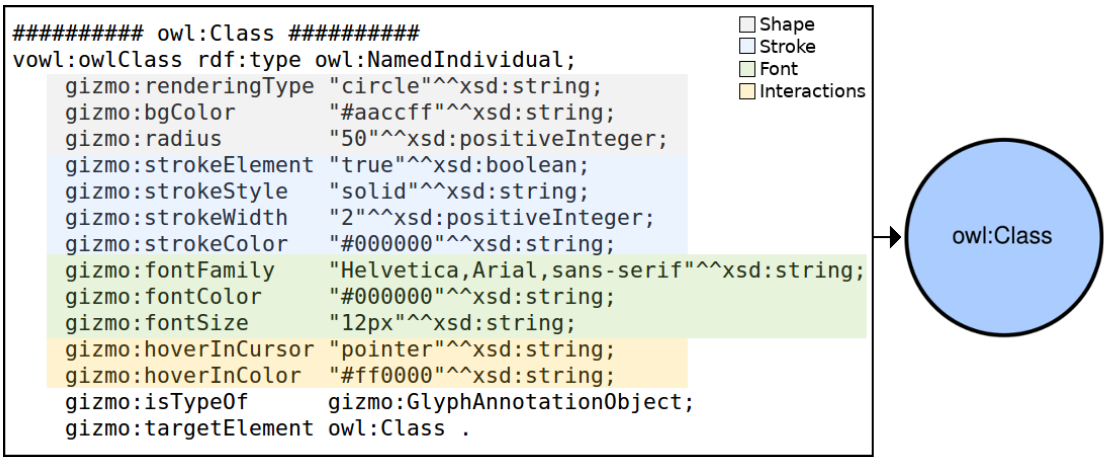
GizMO property	**Description**
Linking Properties	
`targetElement`	Linking visual properties to OWL constructs.
`subjectElement`, `predicateElement`, `objectElement`	Graph triple pattern definition.
`subjectDescription`, `predicateDescription`, `objectDescription`	Definition of visual properties for corresponding rendering primitives in views.
Visual Properties	
`renderingType`	Specification of geometric shapes.
`width, height, radius`	Specification of geometric shape characteristics.
`bgColor, strokeElement, strokeWidth`	Specification of visual property characteristics.
`position_x, position_y, visible`	Spatial information and visibility status.
`link_strokeStyle, link_arrowHead_renderingType`	Specifications for the visual appearance of links.
Annotation Objects	
`annotationObjectDescription`	Description for its purpose.
`isTypeOf`	Specification into glyph, visualization, triple, notation, and view annotation object.

[1]https://github.com/gizmo-vis/gizmo/blob/master/coreOntology/gizmoCore.ttl

The methodology employs the separation of concerns paradigm for the representation model. Thus, visual depictions are combinations of the associated annotation ontologies (i.e., notations and views). The methodology creates these associations using `owl:imports` statements. However, defining visual representations for ontologies requires the *authorship* of the ontology. Authors of ontologies can explicitly import notations and views into an ontology and publish it with the desired visualization. In contrast, containers enable to associate the domain ontology with its visual representation if no authorship is available using the import statements for each of the required ontologies (i.e., domain ontology, notation(s), and view(s)).

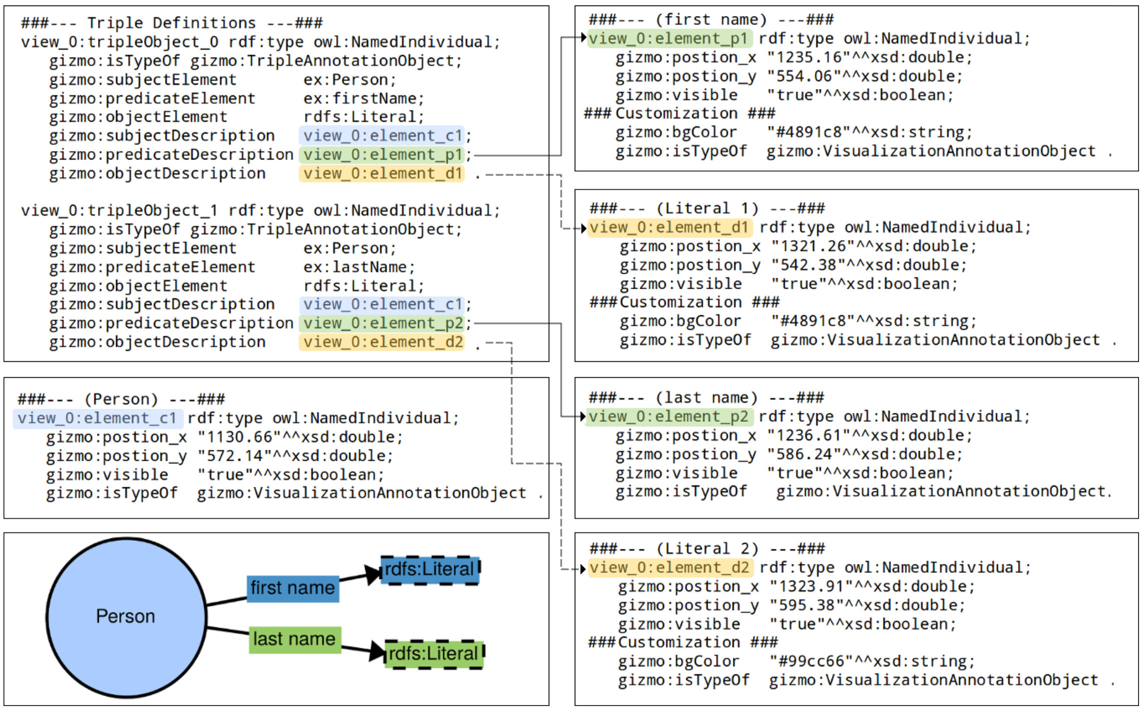
Our technical realization builds upon the assumption that any ontology visualization tool has a parsing mechanism. Ontology parsers, such as the OWL-API, load an ontology and its import statements into a combined model. However, in this combined model, identifying elements from the domain ontology (also optional other imported domain ontologies) and the visualization resources is necessary. Since definitions for visual depiction are grouped in named individuals that use the annotation properties of the GizMO core ontology, our implementation uses its namespace to distinguish between conceptual and visual elements. Named individuals of different annotation ontologies are disambiguated by their namespaces and annotation object types, e.g., the type *glyph annotation object* corresponds to a named individual of a notation. Figures 5.3 and 5.4 show examples of instantiated named individuals for notations and views.

Since OWL does not provide a conceptualization of an "annotation object" that groups annotation properties, our realization of the GizMO core ontology uses a simplified assertion of visual properties. In particular, we use XML Schema datatypes (xsd) for assigning values to visual properties. Visual attributes are annotation properties that are grouped in instances solely of the type `owl:NamedIndividual`. These instances become automatic instances of `owl:Thing`, which are on a higher abstraction layer compared to elements from the domain ontology. Thus, for the visualization of ontologies, such a simplified assertion has a practical advantage: GizMO does not introduce a single class. This has the benefit that our representation model will not conflict with other visualization tools. For example, the class tree of Protégé will not be cluttered with elements that correspond to the definitions for the visual representation.



```
########## owl:Class ##########
vowl:owlClass rdf:type owl:NamedIndividual;
    gizmo:renderingType "circle"^^xsd:string;
    gizmo:bgColor       "#aaccff"^^xsd:string;
    gizmo:radius        "50"^^xsd:positiveInteger;
    gizmo:strokeElement "true"^^xsd:boolean;
    gizmo:strokeStyle   "solid"^^xsd:string;
    gizmo:strokeWidth   "2"^^xsd:positiveInteger;
    gizmo:strokeColor   "#000000"^^xsd:string;
    gizmo:fontFamily    "Helvetica,Arial,sans-serif"^^xsd:string;
    gizmo:fontColor     "#000000"^^xsd:string;
    gizmo:fontSize      "12px"^^xsd:string;
    gizmo:hoverInCursor "pointer"^^xsd:string;
    gizmo:hoverInColor  "#ff0000"^^xsd:string;
    gizmo:isTypeOf      gizmo:GlyphAnnotationObject;
    gizmo:targetElement owl:Class .
```

Figure 5.3: Named individual for visualization of `owl:Class`.



```
###--- Triple Definitions ---###
view_0:tripleObject_0 rdf:type owl:NamedIndividual;
   gizmo:isTypeOf gizmo:TripleAnnotationObject;
   gizmo:subjectElement       ex:Person;
   gizmo:predicateElement     ex:firstName;
   gizmo:objectElement        rdfs:Literal;
   gizmo:subjectDescription   view_0:element_c1;
   gizmo:predicateDescription view_0:element_p1;
   gizmo:objectDescription    view_0:element_d1 .

view_0:tripleObject_1 rdf:type owl:NamedIndividual;
   gizmo:isTypeOf gizmo:TripleAnnotationObject;
   gizmo:subjectElement       ex:Person;
   gizmo:predicateElement     ex:lastName;
   gizmo:objectElement        rdfs:Literal;
   gizmo:subjectDescription   view_0:element_c1;
   gizmo:predicateDescription view_0:element_p2;
   gizmo:objectDescription    view_0:element_d2 .
```

```
###--- (Person) ---###
view_0:element_c1 rdf:type owl:NamedIndividual;
   gizmo:postion_x "1130.66"^^xsd:double;
   gizmo:postion_y "572.14"^^xsd:double;
   gizmo:visible   "true"^^xsd:boolean;
   gizmo:isTypeOf  gizmo:VisualizationAnnotationObject .
```

```
###--- (first name) ---###
view_0:element_p1 rdf:type owl:NamedIndividual;
   gizmo:postion_x "1235.16"^^xsd:double;
   gizmo:postion_y "554.06"^^xsd:double;
   gizmo:visible   "true"^^xsd:boolean;
### Customization ###
   gizmo:bgColor   "#4891c8"^^xsd:string;
   gizmo:isTypeOf  gizmo:VisualizationAnnotationObject .
```

```
###--- (Literal 1) ---###
view_0:element_d1 rdf:type owl:NamedIndividual;
   gizmo:postion_x "1321.26"^^xsd:double;
   gizmo:postion_y "542.38"^^xsd:double;
   gizmo:visible   "true"^^xsd:boolean;
###Customization ###
   gizmo:bgColor   "#4891c8"^^xsd:string;
   gizmo:isTypeOf  gizmo:VisualizationAnnotationObject .
```

```
###--- (last name) ---###
view_0:element_p2 rdf:type owl:NamedIndividual;
   gizmo:postion_x "1236.61"^^xsd:double;
   gizmo:postion_y "586.24"^^xsd:double;
   gizmo:visible   "true"^^xsd:boolean;
   gizmo:isTypeOf   gizmo:VisualizationAnnotationObject.
```

```
###--- (Literal 2) ---###
view_0:element_d2 rdf:type owl:NamedIndividual;
   gizmo:postion_x "1323.91"^^xsd:double;
   gizmo:postion_y "595.38"^^xsd:double;
   gizmo:visible   "true"^^xsd:boolean;
###Customization ###
   gizmo:bgColor   "#99cc66"^^xsd:string;
   gizmo:isTypeOf  gizmo:VisualizationAnnotationObject .
```

Figure 5.4: Disambiguation for glyph-specific information using triple definitions for linking distinct visual properties to rendering primitives. Additional modifications for some objects (e.g., first name) allow for customizations of glyph specific information.

Views – Views require more refined considerations of the linking property approach. Domain ontologies use OWL constructs, such as `rdfs:subClassOf`, `rdfs:Literal`, etc., multiple times. As discussed in Section 5.3.2, these are typically represented as multiple glyphs. Since multiple glyphs correspond to a single OWL construct, a bijective mapping using *only* a single linking property is not possible (cf. Figure 5.4).

GizMO solves this challenge by using multiple linking properties in the *triple annotation object*. Any ontology can be serialized in a triple format (e.g., N3), such that it consists of a set of subject, predicate, and object triples. Since OWL uses URIs for resource descriptions, a triple itself is unique. A triple annotation object provides three distinct linking properties for subject, predicate, and object elements. Three additional linking properties (e.g., `gizmo:subjectDescription`) point to *visualization annotation objects*. These provide the assigned values for their spatial position, visibility status, and optional glyph modification information. Thus, a unique assertion is ensured, even for multiple uses of identical OWL constructs. Figure 5.4 shows the definition of two *triple annotation objects* and their corresponding *visualization annotation objects* that define positions and optional glyph modifications.

5.4 Applications

The utilization of the methodology, however, can only be achieved when tools and frameworks are extended towards the interpretation of annotation ontologies that define visual representation models. In the following, we briefly describe two applications[2] that are utilizing the GizMO representation model.

Both applications use the same rendering engine and *partially* implement the semantic zooming approach for ontology graphs [70]. These applications use an implicit mapping for a defined subset of OWL constructs and default elements for not mapped elements. Furthermore, the collapse/expand mechanisms for datatypes, datatype properties, and object properties are used to describe nested node visualizations. Whereas the semantic zooming approach removes collapsed elements from the visualization, our frameworks render the collapsed elements with their visual descriptions inside the corresponding node. Notations specify the collapse/expand state *globally* in the notation annotation object.

Notation Editor – The notation editor[3] is designed to remove the textual crafting of notations. It enables users to visually create definitions for the representations of OWL constructs in a WYSIWYG manner. Implemented as a proof of concept prototype, the usability, richness of features, and user experience are not in focus. Created notations are exported as annotations ontologies and can be shared, reused, and integrated using `owl:imports` statements.

[2]Landing Page: https://gizmo-vis.github.io/gizmo/
[3]Notation Editor: https://gizmo-vis.github.io/gizmo/notationEditor/

Visualization Framework – The framework[4] is designed for the visualization of ontologies with the GizMO representation model. Additionally, it provides the means to create views and containers. Domain ontologies, containers, and notations can be loaded independently. The framework has a data processing pipeline and a visualization pipeline. The data processing pipeline reads an ontology and organizes the comprised information into domain ontology, notations, and views (if available). When notations or views are missing, the default notation is used, and the spatial arrangement is created automatically.

The visualization pipeline receives the processed data and creates customizable glyph objects for the elements from the domain ontology. These glyph objects are initialized with a default notation. Based on the loaded notation, these are overwritten with the definition asserted in the glyph annotation objects. The visualization pipeline then continues with view definitions. Based on the loaded view, the information for the position, visibility status, and optional glyph modifications are updated in the corresponding glyphs. Additionally, the view annotation object asserts the viewport configuration (e.g., zoom factor).

The current implementation employs only a force-directed layout algorithm. Other layout algorithms pose an implementation effort and are not contributing to the general aspect of this work. However, users can pause the force-directed layout process, manually align the layout and save it as a view. The pause/play state of the force-directed layout is saved in the view annotation object. We provide an overview of the framework's usage and features on the corresponding landing page[2].

5.5 Chapter Summary

In this chapter, we have presented a methodology for customizable visual representations of ontologies. The central aspect of the methodology is its utilization of OWL for definitions of visual representation models. The methodology separates the visual abstraction into two layers: The *global* layer reflects users' mental model and addresses the customizable visual representation of OWL constructs. The *local* layer addresses the mental map of users and provides the means to customize the spatial arrangement, visibility status, and optional glyph modifications.

The applicability of the methodology is demonstrated through GizMO, a representation model for graph-based visualizations of ontologies. The GizMO core ontology defines annotation properties for visual attributes (e.g., shapes, colors, positions, etc.). Annotation

[4]Visualization Framework: https://gizmo-vis.github.io/gizmo/visualizationFramework/

objects provide grouped instantiations of values linked to OWL constructs and elements of the domain ontology. Different types of annotation objects target various aspects of the visualization. These provide a conceptual separation between the *global* and *local* layers for the visual representation.

Through the use case of GizMO, we show that the customizable visual mapping of OWL constructs poses a particular challenge. GizMO implements a fixed implicit mapping for OWL constructs in the context of a minimal viable product (MVP) prototype. However, customizable visual representation models have to provide definitions for the mapping of OWL constructs, e.g., the merging of nodes linked via the `owl:equivalentClass` property or the multiplication of glyphs. Future work and refinement of GizMO will address the customizable mapping of OWL constructs.

GizMO indicates additional requirements for the disambiguation of multiplied glyphs. OWL constructs, such as `rdfs:Literal`, are used multiple times. Therefore, the challenge of identifying corresponding glyphs in the visualization is introduced. GizMO addresses this challenge through the use of triple annotation objects. These identify the corresponding triples and assert for subject, predicate, and object elements from the domain ontology the corresponding visualization annotation objects. Visualization annotations objects define the spatial information, visibility status, and optional glyph modifications. The visibility flag fosters the reduction of cognitive load by excluding elements from the visualization. The optional glyph modifications enable the customization of visual attributes (e.g., colors and shapes) for elements of the domain ontology.

The design decisions for the methodology and the technical realization of GizMO are conceived to facilitate the customizable definitions for the visual representation of ontologies. The success of the methodology and GizMO depends on the integration into other frameworks and tools. GizMO is currently limited in its coverage of OWL constructs and implicit mappings. Some visual representations cannot merely be described by the visual appearance and spatial arrangement of glyphs. We describe the UML-based notations using collapsing and expanding mechanisms, whereas other notations may require additional behavioral descriptions (e.g., edge bundling). Regardless of its limitations, Figure 5.5 illustrates the variety of already possible visualizations.

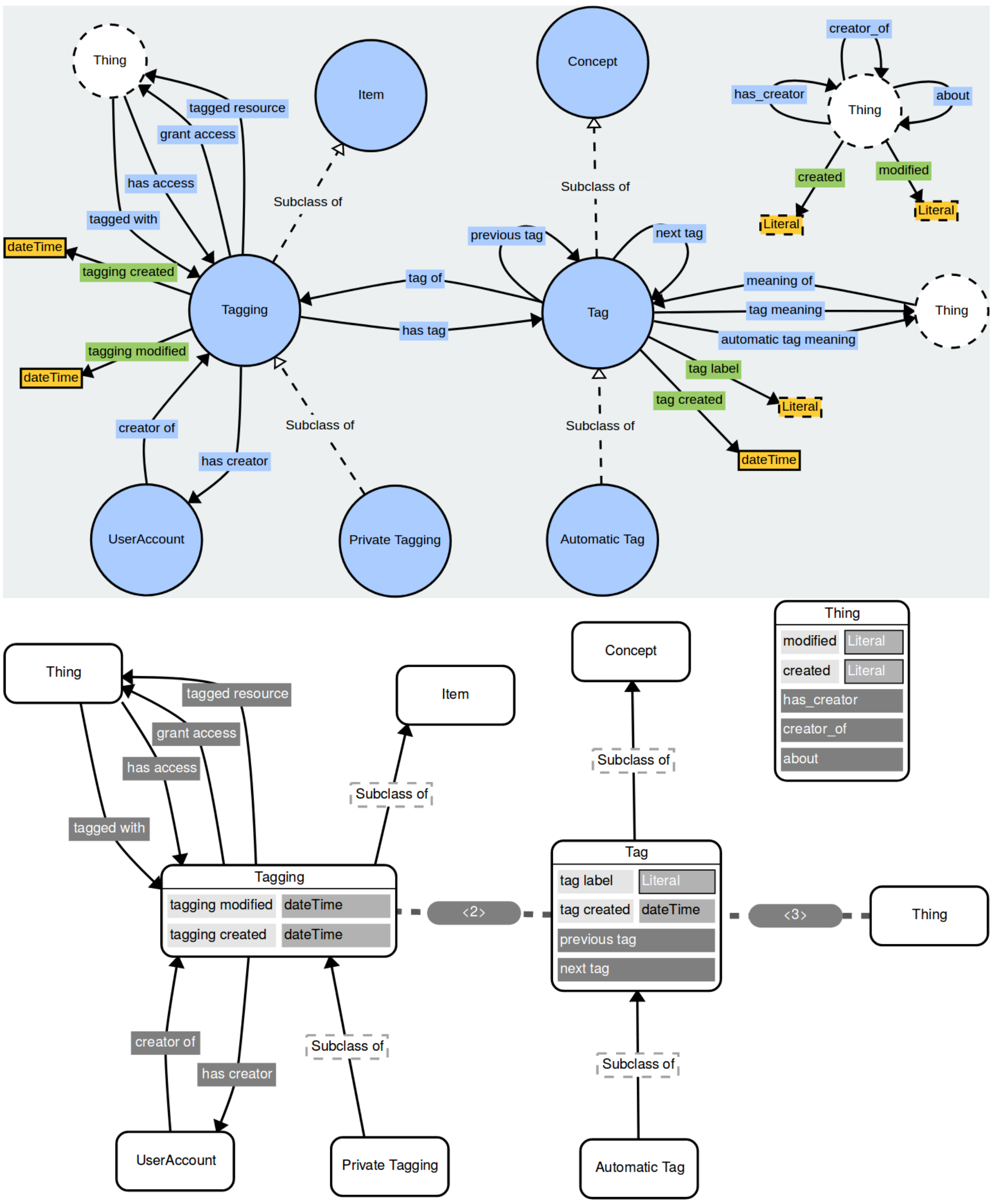


Figure 5.5: Examples created with GizMO, realizing VOWL and UML notations.

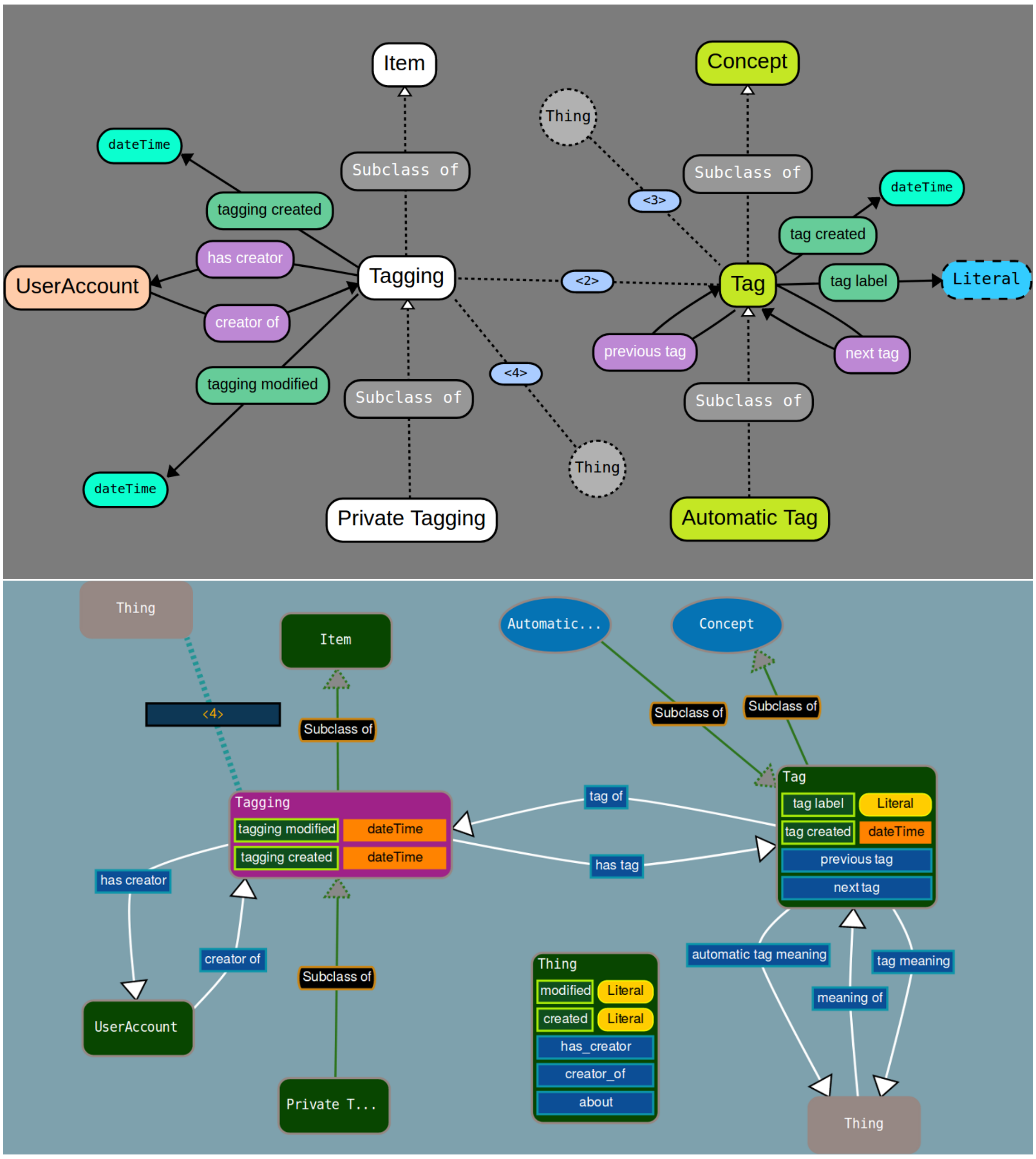


Figure 5.6: Examples created with GizMO, realizing custom notations.

The abstract nature of the methodology and the provided discussion in Subsection 5.2.1 indicate the induced responsibility for realizations, such as GizMO. On the one hand, GizMO has been designed in a minimum viable product (MVP) context, and its limitations are prevalent towards the customizable mappings. On the other hand, evaluating the usability and richness of features will assess the implemented visualization frameworks and not GizMO or the methodology.

Suitable visualizations highly depend on the use case and the targeted user group. The methodology, its realization using GizMO, and corresponding prototype implementations provide evidence to answer **RQ2** that approaches with customization capabilities can increase sense-making of Semantic-Web-Based knowledge structures. Customizations allow users to adjust the visual representations to their current needs. Approaches, such as GizMO, could foster communication between knowledge engineers, domain experts, and other user groups, providing the flexible exchange of visual notations.

CHAPTER 6

Customizable Chart Visualizations of Tabular Data in Knowledge Graphs

Knowledge Graph visualizations in the form of node-link diagrams facilitate understanding of its structure, i.e., resources, their attributes, and interrelations. However, Knowledge Graphs are not limited to schema or vocabulary data. Various works, such as the work of Vu et al. [67], address the transformation of tabular data into Knowledge Graph representations. Tables provide an *organized* and compressed depiction of information and are frequently used in scientific and industrial contexts. The objective of visual representations is to present information so that the user can quickly understand its content. However, node-link diagram visualizations are typically not suitable to represent tabular data. With the growing size of the table, its visualization as a node-link diagram becomes hard to read and analyze due to information overload and cluttered representation. Chart visualizations provide different views on the data, facilitating its understanding and analysis. However, suitable chart visualizations also depend on the use case, the data, and target user groups. Our approach realizes customizable chart visualizations using additional semantics. This chapter continues to address the following research question:

Research Question 2 (RQ2)

How can we improve understanding of Semantic-Web-Based knowledge structures using interactive and user-centered visualizations?

The contributions of this chapter are the following:

- Additional semantics enabling information organization and customizable chart visualization generation
- An approach towards customizable chart visualization for tabular data originating from Knowledge Graphs.

This chapter is based on the following publication: [118]

The remainder of this chapter is structured as follows: Section 6.1 motivates the approach by addressing scholarly communication in the form of scientific articles. Section 6.2 presents an example walk-through for a use case addressing tabular data from scientific articles. Section 6.3 describes the approach and its additional semantics in more detail. Section 6.4 discusses the approach and its capabilities for advanced use cases.

6.1 Motivation

Scholarly communication has not changed in its core during the last centuries. Research articles are typically distributed as PDF documents. Therefore, rendering the extraction and analysis of results a cumbersome, error-prone, and often manual effort. Additionally, the amount of publications increases continuously every year [119]. As a consequence, searching, understanding, and organizing information becomes a burden. Finding and reviewing the literature is tying up cognitive capacity [42] and consumes time, which consequently reduces the time available for original research. The purpose of scientific articles is to inform and share findings. As a means for scholarly communication, the information is presented in documents using text, figures, and tables. While the descriptive text provides detailed insights, figures and tables serve as visual, structured, and compressed representations of information. However, this information is buried in PDF representations [120].

The current developments in scholarly communication make use of Semantic Web technologies. These advancements transform scholarly communication from document-based to knowledge-based information systems employing structured, interlinked, and semantically rich Knowledge Graphs [42]. In contrast to other Digital Library applications that organize primarily bibliographic metadata, the Open Research Knowledge Graph [121] (ORKG[1]) targets to represent the content of research articles as a Knowledge Graph (e.g., research problem, materials, methods, and results).

[1] https://orkg.org

Generally, the view on the information in scientific articles becomes static and frozen following publication. Thus, further analysis of presented information continues to be a manual effort for readers. Knowledge-based representations provide machine-readable access to information, which serves as input for various applications, including those addressing its presentation to humans. Therefore, it is beneficial to extract and transform the information of scientific articles into structured and machine-readable representations. However, due to the design for machine-interoperability and information processing, the cognitive load for humans increases with the growing size and complexity of such data structures. Visualizations serve the purpose of addressing specific information needs for the data at hand. Following the information-seeking mantra (overview, zooming/filtering, and details on demand) [60], we argue that a user-driven approach for the generation of visualizations and their customization can further facilitate the sense-making of information.

In this chapter, we focus on the results of scientific articles that are presented in the form of tables. The objective of our approach is to provide customizable and meaningful chart visualizations of tabular data originating from Knowledge Graphs. In particular, we address the following challenges:

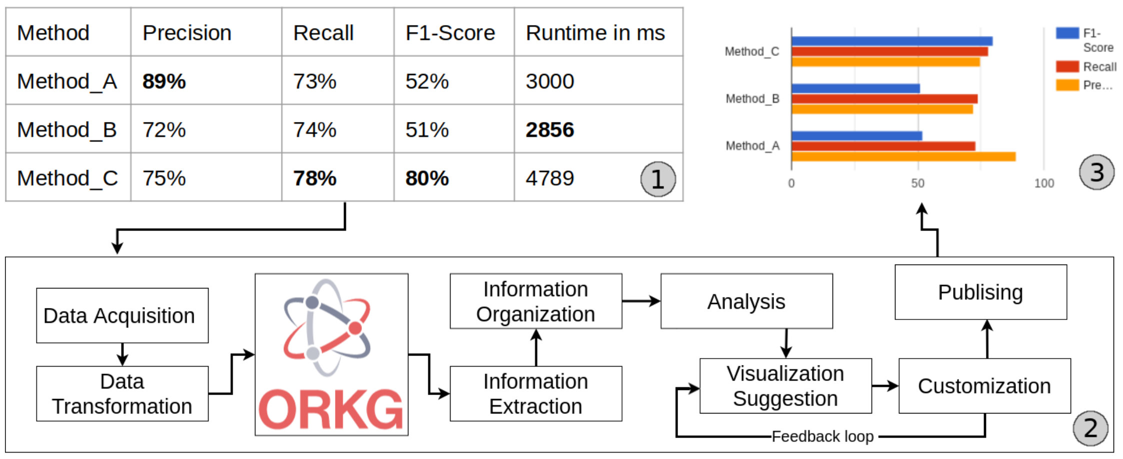
i) What minimal information structure is required in a Knowledge Graph to obtain visual representations of tabular data.

ii) How to analyze this structured data for information organization and customizable chart visualization generation.

Our approach employs a human-in-the-loop technique to transform tabular data into Knowledge Graph representations with additional semantics. These additional semantics serve as the foundation for obtaining views of the data that feed into various visualizations. Using the additional semantics, our approach recreates tables from Knowledge Graphs and enables the analysis of their content to create customizable chart visualizations.

6.2 Exemplary Walk-through

Our approach is motivated and aligned with the objectives of the Open Research Knowledge Graph (ORKG) [121], i.e., the structured representation of contributions in scientific articles and the facilitation of information perception and its sense-making. However, our approach addresses the customizable visualization for tabular data that originates from Knowledge Graphs. As a running example, we use an imaginary table summarizing the performance



Method	Precision	Recall	F1-Score	Runtime in ms
Method_A	**89%**	73%	52%	3000
Method_B	72%	74%	51%	**2856**
Method_C	75%	**78%**	**80%**	4789

Figure 6.1: **Processing Pipeline Overview**: (1) A table for artificial results of Precision, Recall, F1-Score, and Runtime. (2) Processing pipeline. (3) Resulting visual representation.

of different methods, which is common in Computer Science articles (see Figure 6.1).

The table is processed in a pipeline, ensuring that the data is enriched with additional semantics for information extraction, information organization, and analysis. The data transformation process applies a human-in-the-loop approach to ensure the correct assignment of additional semantics. The table is reconstructed using the a-priory known data structure, and its information is organized in groups. The content of information groups is analyzed for the creation of chart visualizations.

6.2.1 Data Acquisition and Transformation

At first, the data acquisition phase transforms the table into a Knowledge Graph representation and ensures the correct assignment of additional semantics. These additional semantics serve as additional context information for the reconstruction of tabular data. Knowledge Graph structures typically reflect a triple-based representation $< s\ p\ o >$, where the predicate p interlinks the subject s and the object o. The Knowledge Graph representation of tabular data highly depends on the transformation model. In this work, we focus on the extraction of information from Knowledge Graphs. However, we require a transformation model for tabular data that fulfills a simple structure for the reconstruction of tables and additional semantics to generate customizable chart visualizations.

A simple row-based transformation model identifies the subjects as row entries in the first column. The headers of all other columns reflect the predicates, and the cell values refer to the objects. The simple transformation process generates the following triples:

```
Subject        Predicate         Object
--------------------------------------
ex:Method_A    ex:hasPrecision   "89%" .
ex:Method_A    ex:hasRecall      "73%" .
ex:Method_A    ex:hasF1-Score    "52%" .
ex:Method_A    ex:hasRuntime     "3000 ms" .
```

Listing 6.1: Simple transformation of tabular data into triples.

While this simple transformation process brings the tabular data into a machine-readable format, the triples merely reassemble data points without any context. Furthermore, Semantic Web technologies allow for creating data in various ways depending on the granularity and structural representation. Thus, to retrieve information from a Knowledge Graph, the user would have to know the underlying semantics for the predicates and objects. Additionally, knowledge about the transformation model's target structure is required to identify the triples addressing the tabular data information.

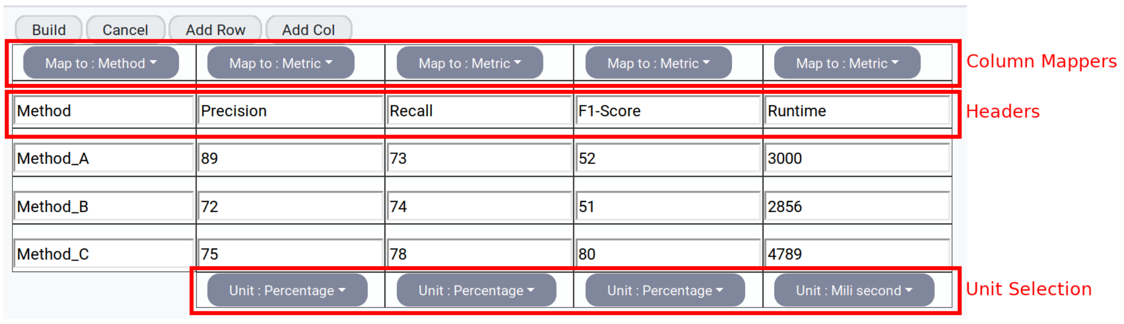
We describe an approach that augments tabular data with additional semantics during the data acquisition phase. The transformation model preserves the *context* and allows for creating further analysis and visualizations from this structured data more efficiently. Maintaining the context requires the augmentation of cell values with additional semantics. Our transformation model builds upon the following heuristics:

i) The first column holds the cell entries for the subjects. In our example, these are the methods. Thus, a row corresponds to a resource that is bound to the cell value of the method. Related to this, our transformation model is also row-based.

ii) Other columns provide values for measurements of a metric. Thus, our transformation model adds to the cell value two additional attributes: the metric and the unit. The header values of the columns determine the metric, while a human-in-the-loop approach assigns the units for the corresponding columns.

While, in general, the particular value is of interest, it is also necessary to incorporate the *context*. The numerical value "89" is just a data point lacking any meaning. Adding a metric and a unit to this value captures more *context*. The value "89" refers to *Precision* and is given in percent. However, this information is not sufficient to make use of it because it does not relate the metrics to a method. Thus, the method serves as an additional context item. The incorporation of the context enables us to describe the cell value as: The value "89" describes *Precision*, it has the unit percentage, and it refers to a method.

Figure 6.2 shows a simple tabular input widget that eases the data entering process for users and ensures the correct assignment of additional semantics for the table. Figure 6.3 shows the result of the transformation in the form of a node-link diagram, indicating an overcrowded visual representation.



Map to : Method	Map to : Metric	Map to : Metric	Map to : Metric	Map to : Metric
Method	Precision	Recall	F1-Score	Runtime
Method_A	89	73	52	3000
Method_B	72	74	51	2856
Method_C	75	78	80	4789
	Unit : Percentage	Unit : Percentage	Unit : Percentage	Unit : Mili second

Figure 6.2: Widget for the tabular data transformation process eases the data input process and appends additional semantics to cell values.

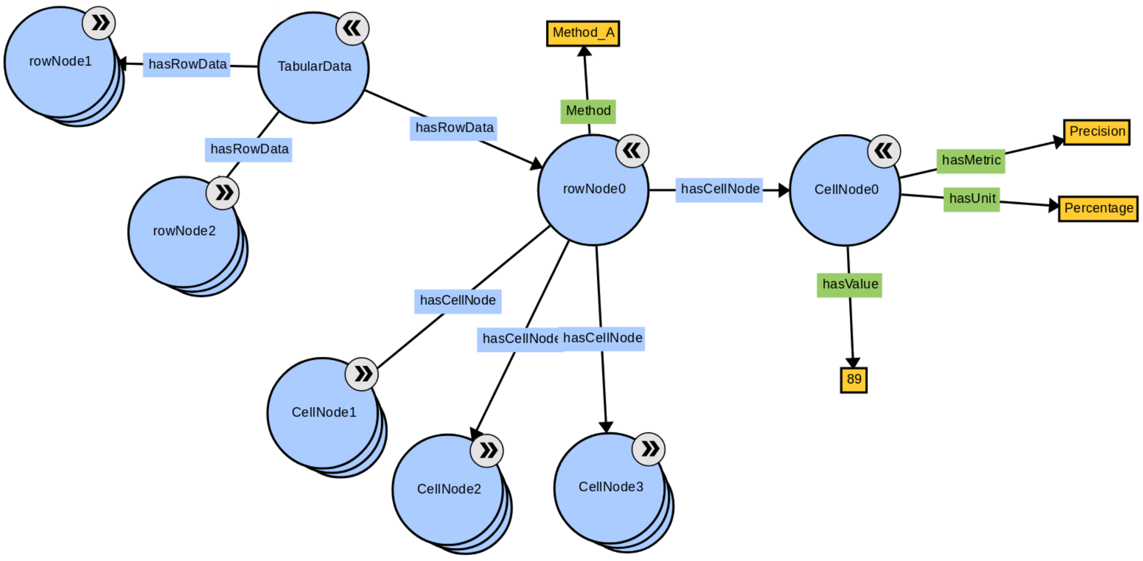
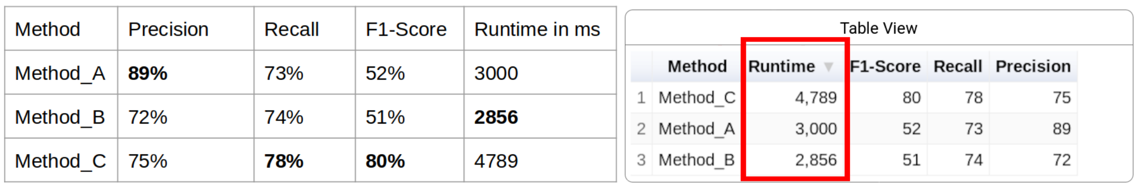


Figure 6.3: Corresponding Knowledge Graph representation illustrated as a node-link diagram. Tabular data is organized in row nodes, which provide the corresponding method. Each row node lists the related cell nodes with additional semantics for units and metrics.
Note: Some nodes are collapsed to reduce visual clutter and information overload.

6.2.2 Customizable Chart Visualizations

The reconstruction of a table requires additional information about the transformation model and its structural representation. This information is obtained from the data acquisition phase that creates the structured representation with additional semantics. However, due to the unknown order of returned triples, the organization of rows and columns can change. Nevertheless, we obtain a reconstructed table with sufficient *context* for our example. Furthermore, the reconstructed table becomes interactive through corresponding implementations, e.g., sorting the columns in ascending or descending order based on their values. As illustrated in Figure 6.4, this straight forth and back transformations provide already interactions with tabular data and another view on the information.



Method	Precision	Recall	F1-Score	Runtime in ms
Method_A	**89%**	73%	52%	3000
Method_B	72%	74%	51%	**2856**
Method_C	75%	**78%**	**80%**	4789

Table View

	Method	Runtime ▼	F1-Score	Recall	Precision
1	Method_C	4,789	80	78	75
2	Method_A	3,000	52	73	89
3	Method_B	2,856	51	74	72

Figure 6.4: Illustration of the original table and the reconstructed table from a Knowledge Graph. Runtime is sorted in descending order and highlighted by a red box. *Note*: The ordering of the columns is not preserved.

The reconstructed table serves as input data for chart visualizations. However, the data units are a crucial factor in creating meaningful chart visualizations. As illustrated in Figure 6.5, a column chart visualization represents metrics *F1-Score* and *Runtime*. Due to the significant differences in the data ranges, the chart visualization provides a false impression. It shifts the attention focus to the visual elements that have a higher presence in the chart. We argue that metrics with the same units provide *reasonable* candidates for grouping information and avoid false interpretations when visualized in the same chart.

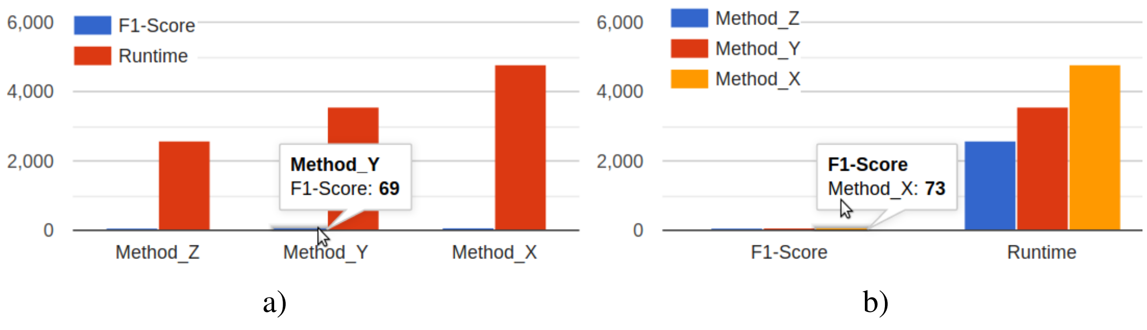
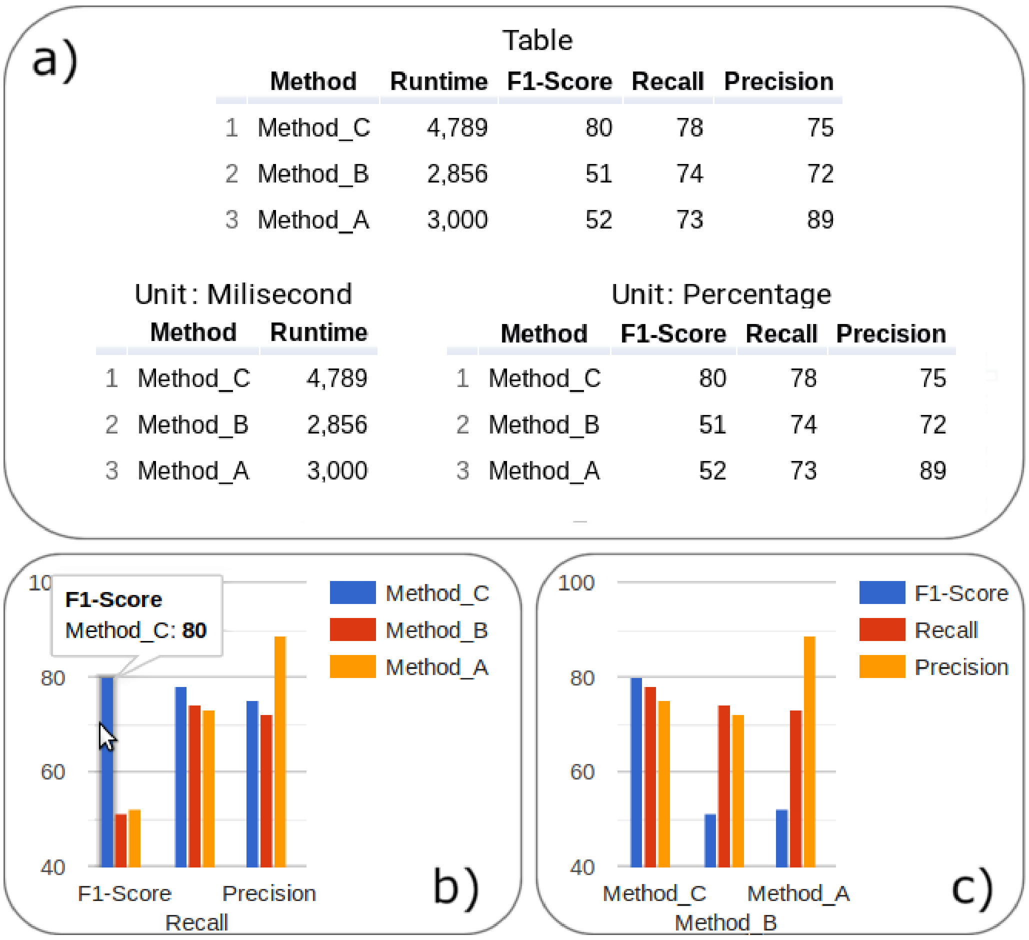


Figure 6.5: Column chart visualization indicating the possible false first impression through unrelated units and large differences in the data ranges.

The semantics of metrics enable additional rules for selecting suitable visualizations from the space of possible chart types. As a negative example for the visualization of *Precision*, we classify the pie chart and box-plot visualizations as not suitable because they are typically used to represent (statistical) distribution values. Once the analysis step has provided a set of candidate visual representations, the user has the option to select a chart type. Furthermore, the user has the option to adjust the mappings for the x-axis and the labels. These options allow organizing the information representation to the user's preference. Our implementation supports the following representation types currently: table, line chart, bar chart, and column chart. As illustrated in Figure 6.6, two different views can increase comprehension by providing the information representation grouped based on the method or the metric.


a)

Table

	Method	Runtime	F1-Score	Recall	Precision
1	Method_C	4,789	80	78	75
2	Method_B	2,856	51	74	72
3	Method_A	3,000	52	73	89

Unit : Milisecond

	Method	Runtime
1	Method_C	4,789
2	Method_B	2,856
3	Method_A	3,000

Unit: Percentage

	Method	F1-Score	Recall	Precision
1	Method_C	80	78	75
2	Method_B	51	74	72
3	Method_A	52	73	89

Figure 6.6: a) Information organization process creates sub-tables based on units;
Two visualizations presenting the information group of Precision, Recall, and F1-Score: b) grouped by metric; and c) grouped by the method.

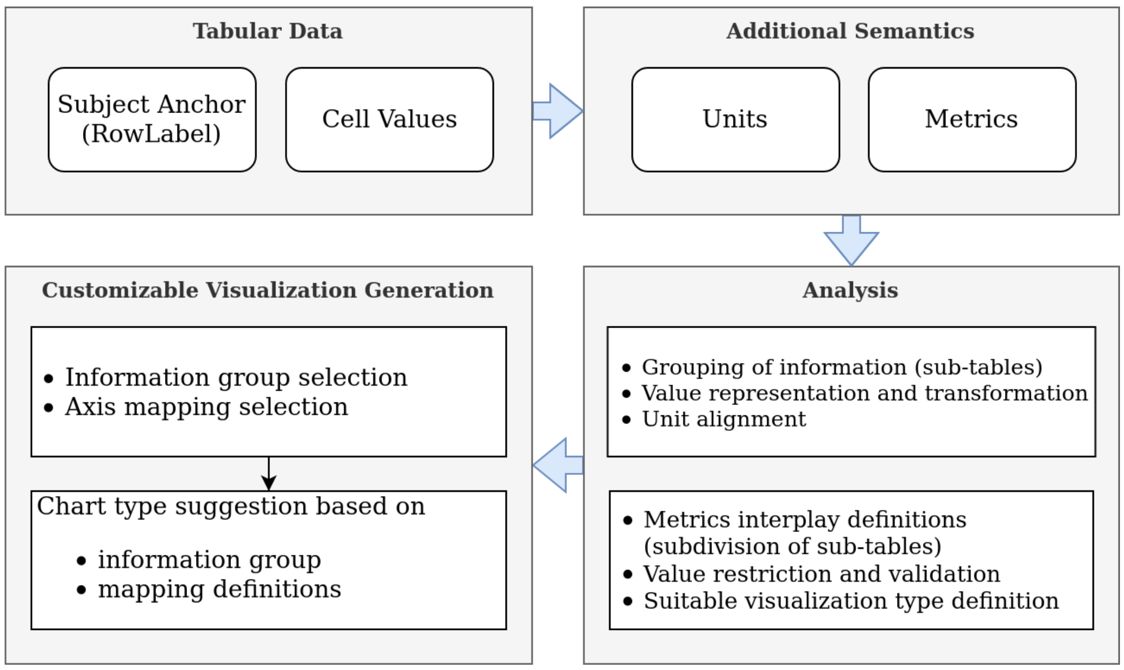

Tabular Data
Subject Anchor (RowLabel)
Cell Values
Additional Semantics
Units
Metrics
Analysis
• Grouping of information (sub-tables)
• Value representation and transformation
• Unit alignment
• Metrics interplay definitions (subdivision of sub-tables)
• Value restriction and validation
• Suitable visualization type definition
Customizable Visualization Generation
• Information group selection
• Axis mapping selection
Chart type suggestion based on
• information group
• mapping definitions

Figure 6.7: Schematic overview of the approach. The additional semantics for units and metrics enable analysis for the generation of customizable chart visualizations.

6.3 Approach

Our approach operates under the assumption that the used data structure converting tables to Knowledge Graph representations is known. We use a simple row-based transformation model to convert one-dimensional tabular data into Knowledge Graph representations to satisfy this assumption (cf. Subsection 6.2.1). While this transformation model enables the reconstruction of tables, our approach defines additional semantics and rules for guiding the generation process of chart visualizations. Figure 6.7 provides a schematic overview of our approach that augments the cell values with additional semantics.

6.3.1 Additional Semantics for Tabular Data Originating from Knowledge Graphs

The row-based transformation model provides the structure required for the reconstruction of tables. While tables provide an organized and compressed representation of information, chart visualizations can additionally facilitate information comprehension. However, suitable visualizations are highly dependent on the underlying data and the preferences of users. The flexible structure of tables allows for incorporating different information (e.g., columns addressing other metrics) within one table. Inadequate representations result from information organization of unrelated metrics (cf. Figure 6.5). Visualizations facilitate the comprehension of information. However, users have different expectations and previous experiences [116]. Consequently, providing customizable chart visualizations can increase sense-making through different views on the information.

The reconstructed table reflects the selection and organization of the corresponding data entries from a Knowledge Graph. However, we argue that the *context* is viable for creating suitable chart visualizations. In this section, we define the context of a cell value as follows:

Definition 6.1: Cell Value Context

Context(value(i,j))=(RowLabel(0,j), Unit(i), Metric(i))
Where $i >= 1$, is the column index and j the row index.

The *RowLabel* refers to the entries from the first column that are used as subjects in the Knowledge Graph representation. A user provides the *Unit*. The *Metric* is obtained from the column header values. Referring to the motivating example, our approach obtains the context for the value 89 as: Context(89) = (Method_A, Percentage, Precision).

The semantics of *Units* provide the means to create information groups by clustering columns, i.e., the extraction of sub-tables through the matching of compatible units. These groups reflect information that relates (or co-relates) to a certain extend. The semantics of *Metrics* provide the means to guide the selection of suitable chart visualization types. In particular, it is the definition of compatible chart types for individual metrics.

Units: The additional semantics of *Units* provide the means to align the cell values to a uniform representation for a particular unit. Cells can have various representations of values, e.g., in the case of percentage, the value can be represented as 89%, 0.89, or .89 and possibly more scientific numerical representations, such as 0.089×10^3. Therefore, sub-tables are analyzed based on their literal values (i.e., the string representation of an entry originating from a Knowledge Graph). Units, such as percentage or milliseconds, referring to numerical values, employ simple regular expression rules to transform the strings to a numerical representation: The first rule clears unit symbols (e.g., % or ms) from the string representation because the units are already assigned in the context of the value. The second rule removes symbols, such as spaces and commas, that are used for visual guiding (e.g., 1000000 can be represented as 1 000 000 or 1,000,000). The third rule detects scientific representations and transforms them into numerical values.

Units specify a target value representation, e.g., percentage expects values in the 100-base. Mismatching numerical representations are identified by analyzing the data range of the columns in a sub-table. Observing the value 0.72 allows for two interpretations: The value refers to 72% or $0.72\% \iff 7.2‰$. The data range within one column provides the context for the representation transformation. If the data range is $[0.72, ..., 0.89]$ then the analysis step can identify the representation as 1-base and transform it to 100-base. If the range is $[0.72, ..., 89]$ then the system identifies the representation as 100-base. While this transformation rule is a simple heuristic, we argue that values addressing fractions of percentage, such as 0.72%, are better depicted using the unit per-mil.

The semantics of *Units* additionally serve for alignment definitions between them. For example, percentage and per-mil are quickly brought into correspondence using an alignment factor of 10, or milliseconds are transformed to seconds using an alignment factor of 1000. The semantics for unit alignment enable the approach to detect compatible units and bring them into correspondence for clustering related information in sub-tables.

Metrics: The semantics of metrics provide additional criteria for building information groups. As mentioned before, units provide *reasonable* candidates for clustering related information into groups. However, identical units are used in different metrics. For example, percentage can refer to performance measurements in information retrieval tasks or statistical distributions. The definition of compatible metrics refines the grouping of related information and determines which columns serve as input for chart visualizations.

The semantics of metrics provide additional value validation mechanisms. In particular, individual metrics define a data range. For example, the metric *Precision* has a range of $[0, ..., 100]$, or *Runtime* cannot be expressed as negative values. Furthermore, chemical substances that provide the pH-value have the range of $[0, ..., 14]$. These value range restrictions define a validation mechanism for transformation models that populate Knowledge Graphs with tabular data. However, the value range restrictions for the myriad of measurement factors need to be defined individually for each metric.

The semantics of metrics provide additional means for selecting suitable visualization candidates from a larger space of possible chart types. Each metric defines compatible chart visualization types. For example, *Precision* defines bar chart and column chart as compatible types. Other chart types, such as pie charts or box plots, are categorized as incompatible because they are designed for the visualization of statistical distributions. We argue that pie charts are incompatible due to the aspect of *Precision* being a performance metric, and the intention of the visualization is to bring the individual column entries into a representation that is easy to digest and compare. Pie charts illustrate fractions of a whole. Therefore, combining the values for the different methods into one pie is not applicable. Multiple pie charts are required for individual cell values. Related to this, comparing and spotting small differences (e.g., 72% vs. 73%) becomes more challenging in such visual representation. Box plots are incompatible because they are designed to convey the distribution of an observed variable. While this visualization method is applicable for the representation of *Precision*, we argue that the comparison of the values is of interest, i.e., which method performs best and not their distribution. Furthermore, line charts are incompatible because they are designed to describe the change of a variable over time.

6.3.2 Visualization Suggestion

The analysis of the additional semantics performs most of the heavy lifting. The organized and subdivided tables serve as input data for the chart generation. The semantics of the metrics provide a selection of suitable chart types. However, the dimensions of the table also pose restrictions on the selection of suitable chart types. For example, radial charts require at least three dimensions to span an area for a value. While this criterion is met when the number of rows is adequate (e.g., visualizing *Precision* with methods as the axial dimension), this representation becomes invalid if the axis mapping is flipped and the dimensional criterion is not met (e.g., only *Precision* serves as the axial dimension). This example indicates that the selection for axis mapping is also crucial for the visualization suggestion. As shown in Figure 6.1, this refers to the feedback loop for the visualization suggestion. Due to the aspect that our approach currently supports one-dimensional cell values, the customizable mappings reflect the exchange of x-axis and label definitions.

6.4 Discussion of the Approach

Scientific articles often present results in the form of tables. Tables provide an organized and compressed representation of information. However, chart visualizations can additionally facilitate the comprehension of information. Our approach focuses on the visualization of tabular data that originates from Knowledge Graph infrastructures. Additional semantics guide the selection of related information and the creation of suitable chart visualizations.

6.4.1 Limitations

Our approach builds upon the semantics and the structure of tabular data representation in a Knowledge Graph. Thus, it is currently limited to the chosen transformation model that enables the analysis for the chart generation. Our transformation model requires the first column of the table to serve as an anchor for the identification of subjects. Furthermore, the approach addresses the one-dimensional representation of columns and rows.

In our approach, the first column of the table refers to unsorted entries. Entries are placed arbitrarily on an axis without consequences on the perception of the information in a chart. However, when dealing with order-dependent entries, such as time series or physical distances, the position on the axis is significant for information comprehension. Currently, our approach does not address order-dependent entries in the first column.

The current implementation of the approach manages the analysis of the additional semantics using string comparison and ad-hoc rules. The objective of the approach is to enable customizable visualization for tabular data. Thus, we focused on the minimal requirements for the additional semantics. Despite its limitations, our approach brings tables of scientific articles to "life" and enables different views on their information.

6.4.2 Implications and Advanced Use Cases

The approach has been described in the context of tabular data visualizations within a single paper. However, tables are frequently used in scientific articles of various types. Incorporating additional semantics enables new opportunities for analysis of information across papers, too. The information distributed across several tables (in different articles) can be organized for further analysis using the additional semantics.

In the following, we discuss a prototype for an advanced use case in the context of ORKG. Our synthetic data consist of three articles, each reporting results in the form of a table. The three tables are merged into a unified table using ORKG and the additional semantics for units and metrics. The analysis of this unified table follows the same information organization paradigms as described in Section 6.3.

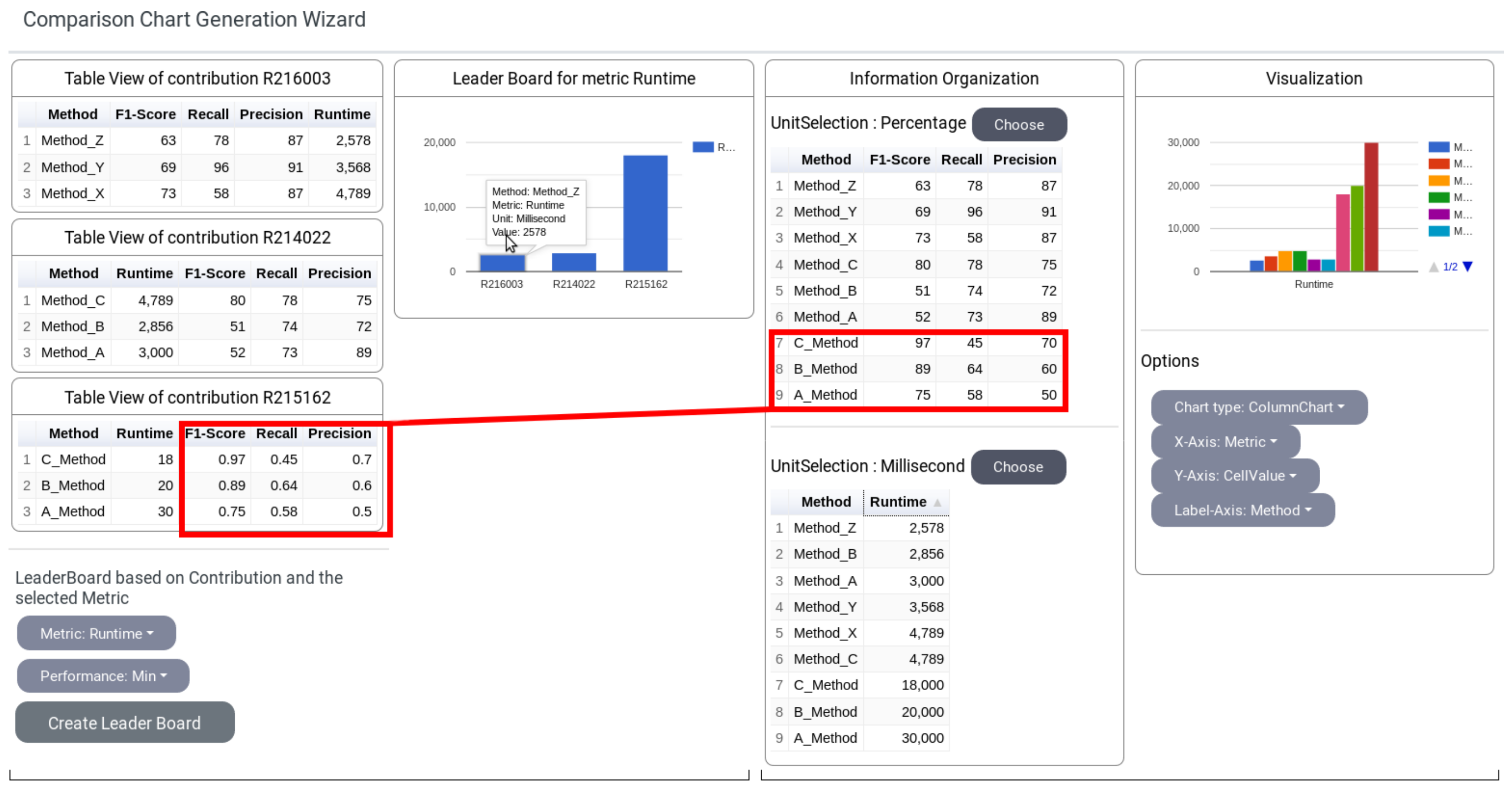
Another application type is a leader-board visualization. Such visualizations, prevalent in Papers With Code[2] and comparable state-of-the-art visualizations, typically address one metric across several contributions. Our prototype provides the user with the selection of identified metrics inside tables. Furthermore, we provide a performance operator

[2]https://www.paperswithcode.com/

selection that determines a single value for the selected metric across all tables. Currently, the prototype implements *min* and *max* performance operators. For example, the *min* operator obtains the best performance for the metric *Runtime*, while *Precision* requires the *max* operator to get the best performing results. Figure 6.8 illustrates the combined user interface for both applications. Additionally, we restrict the leader-board visualization to a column chart. Trend-line (combined scatter plot and line chart) visualizations require a time-dependent variable (e.g., publication date). Thus, an article's publication date has to serve as an additional *context* item for the generation of leader-board charts. While our use case shows the comparison of an example dataset, it neglects the aspect that only related things should be compared. An additional *context* item has to ensure that only compatible tables are merged. As our examples address the performance metrics in NLP-related tasks, the evaluation dataset has to serve as the *context* for identifying compatible tables.

6.5 Chapter Summary

In this chapter, we have presented an approach for customizable chart visualizations of tabular data originating from Knowledge Graphs. A basic transformation model creates Knowledge Graph representations of tabular data. A human-in-the-loop approach is employed in the data acquisition phase in order to assign additional semantics to cell values of the table for the structured and semantically enriched representation. The additional semantics describe metrics and units for the numerical values of tabular data. The semantics of units provide the means to create sub-tables of compatible units and align the numerical values to a uniform representation (e.g., seconds and milliseconds). The semantics of metrics provide the means to create sub-tables for different tables (e.g., merging information from various papers) and validation mechanisms for value range restrictions (e.g., *Precision* is in the range of [0, ..., 100]). Additionally, these semantics define suitable chart visualization methods that are compatible with a *Metric*. A prototype implementation allows users to select a chart visualization from a set of suggested types and customize the axis mappings, providing different perspectives on the information in tables. The visualization suggestion considers the selected axis mapping additionally, ensuring valid recommendations for chart visualization types (e.g., radial-charts require at least three dimensions to span an area to show the information). Using the paper comparison feature of ORKG [122], the approach realizes advanced use cases, such as the visualization of information distributed among tables in multiple articles and leader-boards.



Comparison Chart Generation Wizard

Table View of contribution R216003

	Method	F1-Score	Recall	Precision	Runtime
1	Method_Z	63	78	87	2,578
2	Method_Y	69	96	91	3,568
3	Method_X	73	58	87	4,789

Table View of contribution R214022

	Method	Runtime	F1-Score	Recall	Precision
1	Method_C	4,789	80	78	75
2	Method_B	2,856	51	74	72
3	Method_A	3,000	52	73	89

Table View of contribution R215162

	Method	Runtime	F1-Score	Recall	Precision
1	C_Method	18	0.97	0.45	0.7
2	B_Method	20	0.89	0.64	0.6
3	A_Method	30	0.75	0.58	0.5

LeaderBoard based on Contribution and the selected Metric

Metric: Runtime

Performance: Min

Create Leader Board

Leader Board for metric Runtime

Information Organization

UnitSelection : Percentage Choose

	Method	F1-Score	Recall	Precision
1	Method_Z	63	78	87
2	Method_Y	69	96	91
3	Method_X	73	58	87
4	Method_C	80	78	75
5	Method_B	51	74	72
6	Method_A	52	73	89
7	C_Method	97	45	70
8	B_Method	89	64	60
9	A_Method	75	58	50

UnitSelection : Millisecond Choose

	Method	Runtime
1	Method_Z	2,578
2	Method_B	2,856
3	Method_A	3,000
4	Method_Y	3,568
5	Method_X	4,789
6	Method_C	4,789
7	C_Method	18,000
8	B_Method	20,000
9	A_Method	30,000

Visualization

Options

Chart type: ColumnChart

X-Axis: Metric

Y-Axis: CellValue

Label-Axis: Method

a) b)

Figure 6.8: Chart visualization using the comparison feature of ORKG: a) The individual tables, selection options for leader-board generation, and a leader-board visualization; b) Information organization for merged tables and the resulting column chart. The value representation transformation is indicated in red.

The *context* plays an essential role in extracting tabular data from Knowledge Graphs and creating visual representations. Our approach creates the context using the a-priory known data structure and its additional semantics. Future work will address the extension for the definition of additional semantics related to order-dependent entries for the first column. The semantics of *Metrics* define the interplay among them and which chart visualizations are suitable.

Suitable visualizations highly depend on the use case, the data, and the targeted user group. The approach and its prototype implementation provide evidence to answer **RQ2**. The understanding of the underlying data in Knowledge Graphs can be facilitated using customizable chart visualization in order to provide different views on the data which serve particular information needs for different user groups.

In summary, we argue that the approach introducing additional semantics and further rules will foster the creation of suitable and custom visual representations for tabular data using Knowledge Graphs. Furthermore, it facilitates comprehension through different perspectives on the information in tables. In the following chapter, we investigate how to overcome the limitations of the presented approaches in Chapters 5 and 6.

CHAPTER 7

Customizable Pipelines for Knowledge Graph Visualizations

In Semantic Web contexts, various visualization methods and tools exist. These are typically tailor-suited for specific use cases and tasks. Semantic-Web-Based knowledge structures are realized in a versatile technology stack addressing different use cases. Ontology files are encoded in different serialization formats such as Turtle and N3. Knowledge Graphs use various technologies to store and retrieve information such as SPARQL endpoints, RESTful-APIs with different query languages such as SPARQL, Cypher, Gremlin, and GraphQL. Thus, visual representations of Semantic-Web-Based knowledge structures depend not only on the requirements for the visualization but also on the originating source and how the data is accessed and transformed.

Existing solutions often require modifications to meet the demands of other use cases and different user groups. The work of Dudáš et al. indicates that "*new visualization methods and tools are often developed from scratch, omitting opportunities to learn from previous mistakes or to reuse advanced techniques provided by other researchers and developers*" [17, pp. 1-2].

This chapter addresses the following research question:

Research Question 3 (RQ3)

How can we ease the creation of visual representations in Semantic Web contexts for different use cases and diverse audiences?

The contributions of this chapter are the following:

- Refinement of the visualization generation process.
- Modular, pipeline-based visualization generation framework.
- Visual pipeline builder, providing source code infrastructure as React application.

This chapter is based on the following publication: [123]

The remainder of this chapter is structured as follows: Section 7.1 presents a pipeline-based visualization approach for Semantic Web data. Section 7.2 discusses a prototype realization of the approach. Finally, Section 7.3 summarizes the achieved results under the aspect of the research question **RQ3**.

7.1 Pipeline-Based Visualization Approach for the Semantic Web

Pipeline-based approaches are often used in applications addressing data visualization. Components process a provided input and generate an output that is distributed to other components as input. The pipeline is organized by connections between components distributing input and output data to corresponding elements. This conceptualization allows for creating various visualizations by exchanging components for individual use cases and data sources. Thus, the visualization pipeline approach is flexible and extendable. Accordingly, our approach applies this conceptualization and creates customizable components that are organized in a pipeline.

In this chapter, we propose an approach towards a unified visualization framework for Semantic Web data. Our approach builds on the identification of commonly used steps in the creation process of visualizations. Furthermore, the approach employs separation of concerns paradigms for individual steps, increasing its flexibility. Our framework realizes the approach and provides a modular and customizable visualization pipeline that serves as a foundation for creating visual representations for different data sources, use cases, and user groups in Semantic Web contexts. Customizable components serve as stand-alone artifacts for different steps in the creation process of visualizations. The framework provides the organization of various components in a pipeline, allowing for the creation and selection of *the right components for the right task*, realizing a variety of use cases and visual representations. We present the applicability of the approach using prototype implementations for different components.

7.1.1 Approach

Most visualizations employ the following steps to create visual representations: **i)** access the data; **ii)** map it to visual primitives (e.g., geometric shapes); **iii)** render the primitives; and **iv)** optionally add animations and user interactions. Typically, these steps are created within an application addressing a single use case and a single user group. To address different use cases in the context of Semantic-Web-Based knowledge structures, flexible, customizable, and extendable approaches are required. Our approach refines the visualization generation steps using the separation of concern paradigm. The work of Dudáš et al. [17] indicates that most methods and tools visualize the content of ontologies using two-dimensional graph-based representations in the form of node-link diagrams. Therefore, our approach focuses on the flexible and customizable realizations of visual representations in the form of node-link diagrams. Individual steps are further separated into smaller steps, targeting the flexible realization of different use cases in Semantic Web contexts. The refined steps are accompanied by their corresponding components. Table 7.1 presents the individual steps and their responsibilities.

Table 7.1: Tabular representations of different components and their responsibilities.

Step	Component	Responsibility
1	Data Access Handler	Specify data source and parameters for data retrieval in the JSON format.
2	Parser	Create the `Resource-Relation Model`.
3	Vertex-Edge Mapper	Select data for visualization and create graph structure.
4	Node-Link Mapper	Modify the graph structure for the visualization.
5	Rendering	Create the visual primitives and orchestrate the other components (6 – 8).
6	Visual Appearance	Specify how of nodes and links are rendered.
7	Spatial Layout	Specify how elements are organized in the layout.
8	Interactions	Specify interactions for graph, nodes, and links.

7.1.2 Refining the Data Access

Visualizations typically start with accessing the data **i)**, which is then mapped to visual primitives **ii)**. This observation poses the two following questions: a) What is the data, and how is it accessed? b) How is it mapped to visual primitives?

Semantic Web data comes from different sources and also in various formats. The data sources range from ontology files to Knowledge Graphs that are accessed via SPARQL queries, RESTful-API requests, or other query languages. Thus, individual use cases require specialized mechanisms for accessing data from different sources. Our approach addresses this challenge through the use of the separation of concern paradigm. Accessing data from different sources has the following requirements: First, describe how the data is accessed and fetch it. Second, provide the data in such a way that it can be mapped to visual primitives. Accordingly, our approach refines data access **i)** into two customizable components (i.e., the `Data Access Handler` and a `Parser Component`), addressing each task individually.

Generally, Semantic Web data is accessed using internet technologies. Thus, the customization of the `Data Access Handler` component targets the specification of data sources, such as ontology URIs, SPARQL endpoints, or RESTful-APIs. Additionally, the `Data Access Handler` specifies further parameters for accessing portions of the data source, e.g., the SPARQL query or RESTful-API requests.

Recalling question a), the `Data Access Handler` describes how the data is accessed without addressing the aspect of what the data actually is. In Semantic Web contexts, the main characteristic of data is its textual representation format. It is due to the nature of how Semantic Web technologies encode knowledge representations, e.g., RDF or OWL, with different serializations such as Turtle or N3. Furthermore, SPARQL and RESTful-API results are typically transmitted using textual representations (e.g., XML or JSON formats). Thus, the common factor for different data sources is that their retrieved data is a structured textual representation of information.

The structured and machine-readable textual representation requires a parsing mechanism in order to provide the data in a format that can be mapped to visual primitives. Due to the diversity of different serializations and data structure formats, a unified parsing mechanism cannot be realized. The retrieved textual data representation depends on its originating data source that specifies the resulting format and how the data is organized. However, most modern applications provide the means to request the results in the JavaScript Object Notation (JSON) format. Thus, a form of unification is provided by restricting the output of the `Data Access Handler` to a JSON format.

While having a unified representation of results, the content of the data, and in particular, its structure are governed by the target data sources. Thus, mapping the content to visual primitives requires the a-priory knowledge for the retrieved structure of the JSON results.

Furthermore, these results could hold additional information, not relevant for the visual mapping. Mappings require to filter the data for relevant information for the visualization. Our approach tackles this challenge using customizable parser components, which specify how the JSON structure is transformed into a data model (i.e., `Resource-Relation Model`). Figure 8.1 illustrates a schematic overview for the data access module.

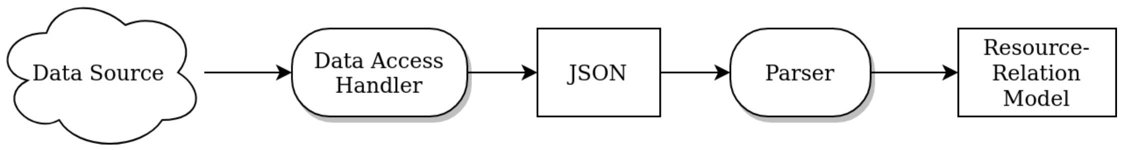


Figure 7.1: Schematic overview for the Data Access Module. The customizable Data Access component specifies the data source and additional information retrieval parameters. The resulting JSON model is processed in the parser component resulting in a `Resource-Relation` model.

7.1.3 Refining the Mapping Process

Typically the mapping process **ii)** transforms the data into visual primitives in a single step. Thus, the visual notation becomes static for the chosen visualization method. However, the objective of our approach is to provide customizable and flexible mappings for a variety of visualization methods. Generally, Semantic Web data reflects a graph structure where resources are connected through relations. Axioms, annotations, constraints, and restrictions are additionally used to define knowledge representations.

Various visual notations employ additional graph manipulations on the network-like structures of Semantic Web data. For example, the VOWL notation [82] performs merge operations on axioms, such as *owl:equivalentClass*, and splitting operations on *rdfs:Literal*s. Nested visualizations, such as UML-based representations, perform nesting operations (e.g., on datatype properties) and visualize these inside the corresponding node.

Our approach uses a two-fold mapping to increase the overall flexibility for different use cases and allow for adjustments using two customizable components. The first mapping transforms the `Resource-Relation` model into a `Vertex-Edge` model. In particular, this component specifies how and if axioms, annotations, or relations are mapped to vertices and edges. Thus, this mapping provides the means for selecting information of interest for the visualization and creates a data model that is processed further. For example, properties such as *rdfs:comment*, *vs:term_status*, or *rdfs:isDefinedBy* are often not displayed as “first-class citizens”. Within a visualization, these can increase information overload and provide overcrowded visual representations. However, use cases targeting particularly such information can be realized by customizing the corresponding mapper to create vertices and edges for these elements. Furthermore, our component does not discard information that is not used in the `Vertex-Edge` model. Vertices and edges are derived from resources and relations. These provide a reference to the corresponding element in the `Resource-Relation` model, allowing for access to this information in later steps.

The second component maps the `Vertex-Edge` model to a `Node-Link` model. In particular, this component specifies merge, split, and nesting mappings. Thus, this mapping provides the means to alternate the resulting graph structure of the `Vertex-Edge` model. Additionally, the mapper allows for creating auxiliary nodes and links that address domain range restriction and relational axioms. Due to the aspect that properties (e.g., *owl:DatatypeProperty* or *owl:ObjectProperty*) are instances of classes [8], these can be displayed as nodes with corresponding links provided as domain and range restrictions. Figure 7.2 illustrates the mapping process for an example dataset. This example emphasizes that the direct transformation of Semantic Web data into a graph requires additional graph manipulations to realize different visual notations.

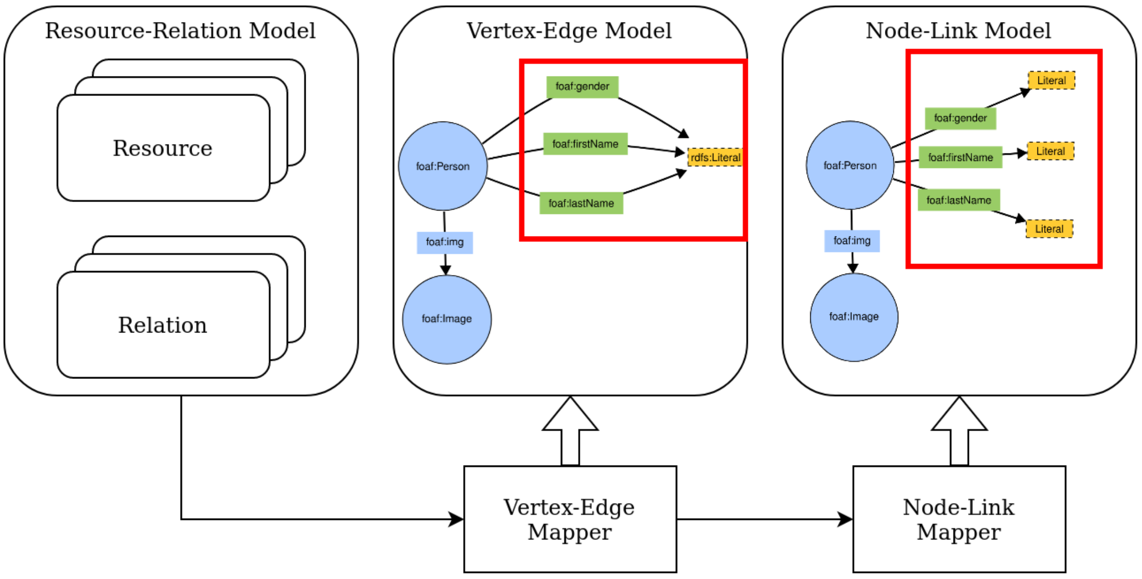


Figure 7.2: The `Resource-Relation` model is transformed into a `Vertex-Edge` model, reflecting a basic graph structure. The `Vertex-Edge` model is transformed into a `Node-Link` model, modifying the graph structure using merge, split, and nesting functions. In this example, split operations are applied to `rdfs:Literal` realizing the VOWL notation. The red boxes highlight the applied split operation.

7.1.4 Refining the Rendering Process

The rendering process **iii)** typically creates a visual representation based on the notation, a layout, and provides user interactions **iv** optionally). Therefore, visualizations become dependent on the realization of the rendering process. However, the objective of our approach is to provide customizable and flexible visual representations that facilitate understanding and interaction with Semantic-Web-Based knowledge structures for different use cases and targeted user groups. Our approach refines the rendering process and optional user interactions into separated steps accompanied by customizable components.

Inspired by GizMO [116], our refinement for the rendering process focuses on node-link diagrams. While GizMO focuses on a methodology for a definition of visual depictions in the form of annotation ontologies, the technical realization used ad-hoc rules to realize different visual notations, such as VOWL or UML, using various graph manipulation operations. GizMO's limitations with respect to ad-hoc graph manipulations are overcome by transforming the responsibility to the customizable mapper components.

In the context of node-link diagram visualizations, the two main visual characteristics of different methods and tools are visual appearance and spatial arrangement of nodes and links. Our customizable components address these aspects individually. The rendering component creates customizable visual primitives for nodes and links. The visual appearance component describes how these visual primitives are rendered. The spatial layout component asserts their position. Additionally, the interaction component specifies user interactions **iv)** for the graph, nodes, and links (e.g., drag, hover, and click interactions).

The customizable `Node-Link` model provides the sets of elements as input for the rendering component. Based on the semantic types of nodes and links, their visual appearance is configurable. Various spatial layout algorithms such as hierarchical trees or force-directed layouts assert the position of elements in the graph.

While GizMO addressed limited user interactions (e.g., mouse hovering) in the context of visual appearance modifications, general interactions such as mouse-click remained non-customizable. We tackle this challenge by creating a customizable interactions component that addresses in the context of node-link diagrams three interaction types: graph, node, and link interactions. Graph interactions specify interactions for the canvas area where visual primitives are rendered. This allows for reusing general interactions such as zooming and panning. Node interactions describe drag, hover, click, and double-click interactions. These interactions are generally bound to all nodes. However, the flexible nature of our components allows for additional modifications using the semantic types of nodes.

Furthermore, additional control elements can be attached to rendered primitives. Link interactions provide the same customizable definitions of interactions. Figure 7.3 illustrates a schematic overview for the individual components of the rendering module.

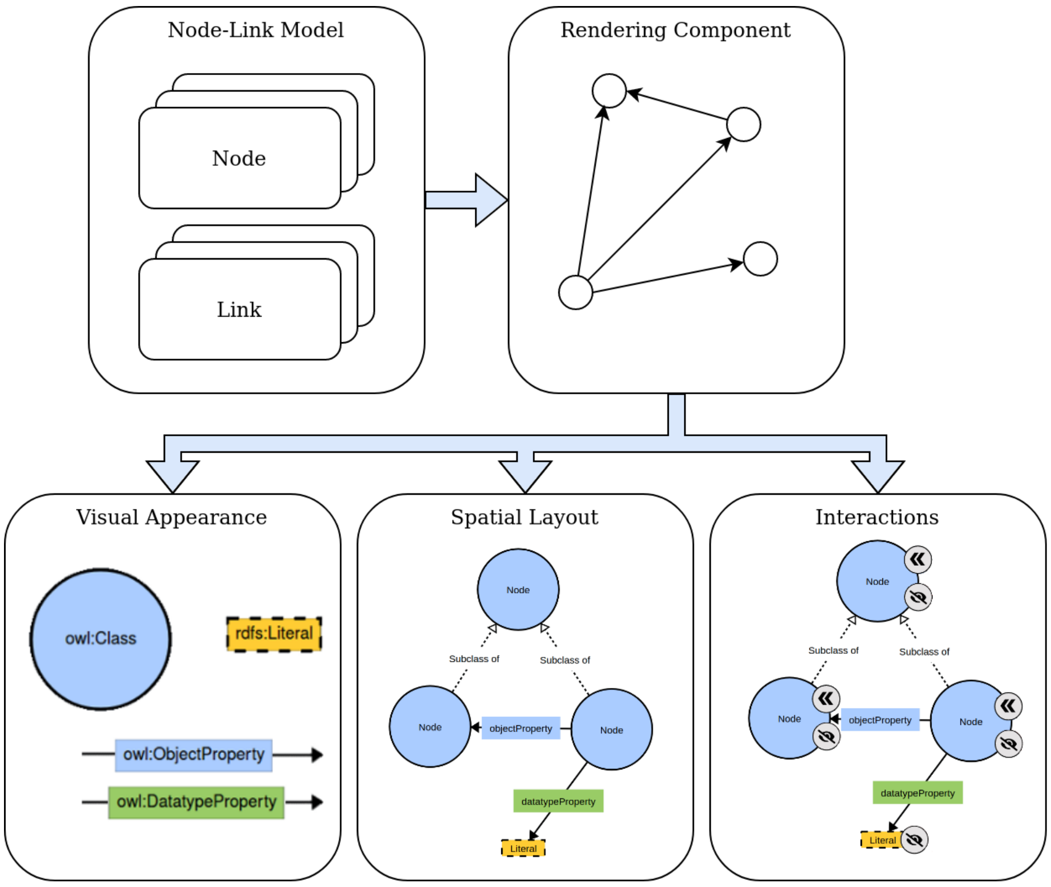


Figure 7.3: A schematic overview of the rendering module. The rendering component creates customizable visual primitives for nodes and links from the `Node-Link` model. The visual appearance and spatial position are asserted in corresponding components. The interactions component creates basic user interactions and provides additional visual primitives for advanced interactions (e.g., node-collapsing to mitigate cognitive load).

7.1.5 Discussion of the Approach

In Semantic Web contexts, various visualization methods and tools exist. However, these are typically tailor-suited for specific use cases and tasks, omitting the opportunity to reuse existing solutions in other use cases. Addressing different visualization use cases in Semantic Web contexts requires flexible, customizable, reusable, and extendable approaches. Our approach refines the individual steps for the visualization generation process, increasing its flexibility for different use cases. Customizable and reusable stand-alone components provide the means to access the data from different sources, create and manipulate graph structures for node-link diagram visualizations. Furthermore, visual appearance, spatial layout, and user interactions are customizable in corresponding components.

The objective of visual representations is to facilitate understanding and interaction with information and data. However, the requirements vary from use case to use case and are also depending on the data at hand. Chart visualizations and other representation methods (e.g., indented trees, chord diagrams, treemaps, and Euler diagrams) facilitate the understanding of Semantic-Web-Based knowledge structures. The limitations of the approach are directed towards the implementation of different visualization methods. While our approach focuses on the flexible realization of node-link diagrams in Semantic Web contexts, its conceptualization as visualization pipelines allows for creating other visual representations. Our components are derived through the separation of concern paradigm. Furthermore, their conceptualization as stand-alone artifacts (i.e., each component takes inputs, processes them, and provides **one** output) allows for their reuse in other visualizations. As our approach targets a unified visualization framework approach, we address this through the divergence of customizable components and convergence in data models. Figure 7.4 provides a schematic overview of unification through divergence conceptualization. The data models, provided as input and output for various components, serve as convergence points within the visualization pipeline. Therefore, we argue that the data models further contribute to the reusability of components in visualization pipelines. Additionally, new data models and components can be created for other visualization methods and use cases.

7.2 Technical Realization and Example Results

In order to evaluate the applicability of the approach, we provide basic implementations for individual components. All data models and the components for the data access module and the mapper module are implemented in plain JavaScript. The rendering components use D3.js additionally for creating interactions and visual primitives as SVG elements. In this section, we describe the technical aspects of individual components and data models. An overview is illustrated in Figure 7.5. Additionally, we showcase the configurations of components for different data sources, namely ontology URIs and SPARQL queries for the DBpedia endpoint. To access various data sources on the Web, we implement a simple proxy that allows us to reduce Cross-Origin Resource Sharing (CORS) restrictions.

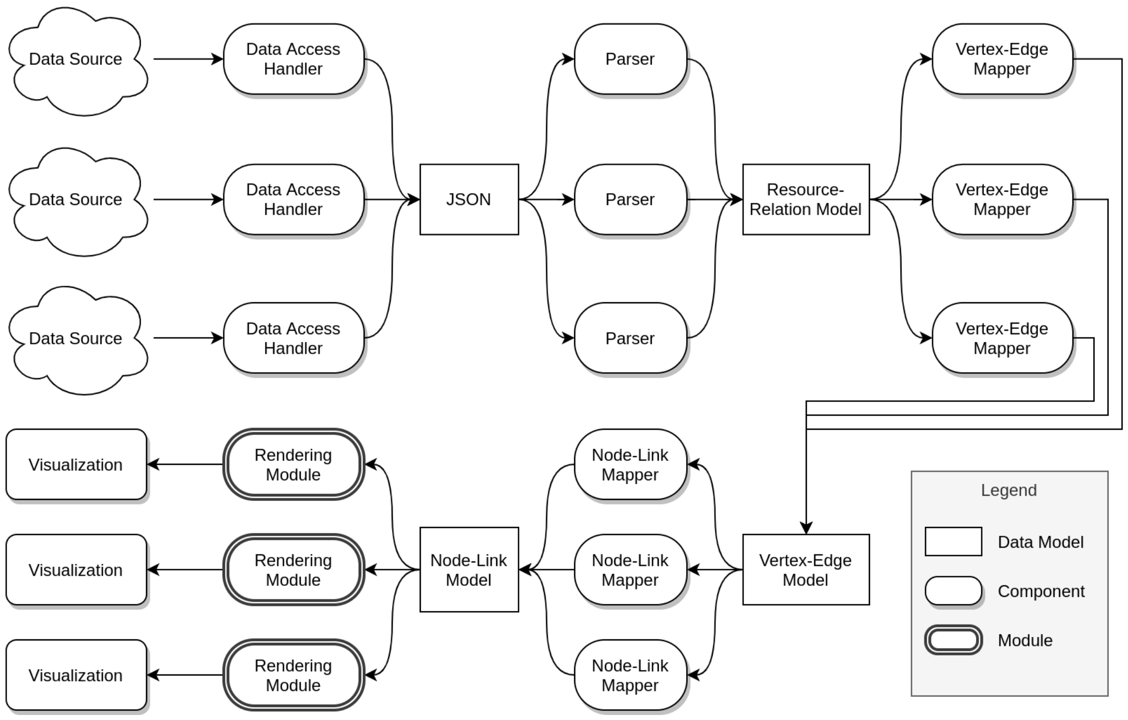


Figure 7.4: A schematic overview for realizing different visualization pipelines indicates the divergence for various components and convergence in corresponding models.

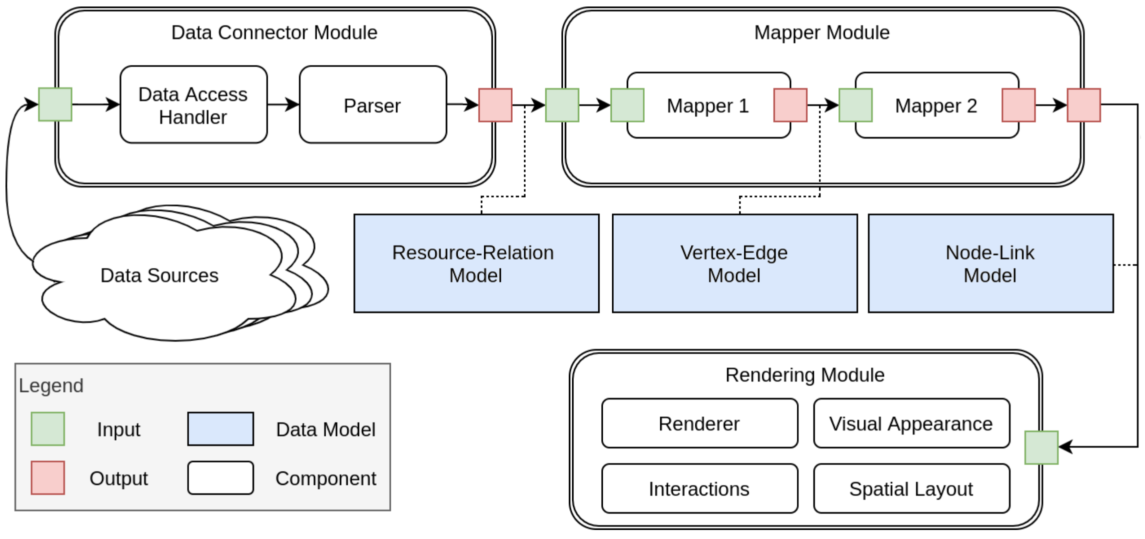


Figure 7.5: An example pipeline using components for creating node-link diagrams.

7.2.1 Modules, Components, and Data Models

In the following, we describe design decisions and technical details for individual components and data models. We define modules as conceptual combinations of components in our terminology, reflecting the typical steps in the visualization generation process (**i–iv**). The definitions for the data models are described alongside the description of the components.

Data Connector Module

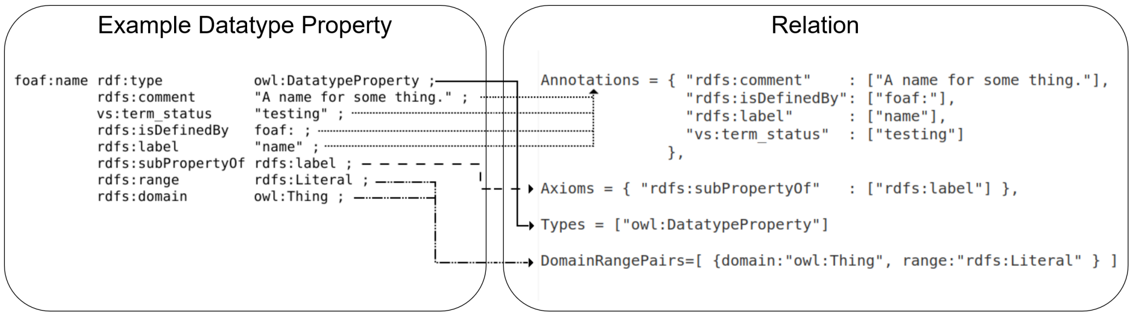
This module realizes access to different data sources. It consists of two components, i.e., the `Data Access Handler` and the `Parser Component`. The output of the `Parser Component` is a `Resource-Relation` model that serves as a foundation for further processing in the pipeline.

Data Access Handler: The `Data Access Handler` accounts for retrieving data from various sources such as ontology URIs, SPARQL queries, and RESTful-API requests. This component specifies the origin of the data and optional parameters for SPARQL queries and RESTful-API requests. Furthermore, this component specifies that its output is provided in the JSON format. While we do not strictly enforce the JSON format, we argue that its availability in most programming languages serves as a reasonable choice.

Parser Component: The `Parser Component` transforms the retrieved JSON data into the `Resource-Relation` model. While the JSON format serves in our approach as a unified representation for data, its structure is a-priory not known. Thus, parser components have to specify how the content of the data is accessed and how it is transformed.

Resource-Relation Model: The `Resource-Relation` model reflects the organization of different elements. Semantic-Web-Based knowledge structures are described using resources, relations, annotations, axioms, type assertions, etc. Accordingly, resources define type assertions, annotations, and axioms. Relations extend resources by providing domain and range restrictions, forming the connection between resources.

The `Resource-Relation` model provides the reorganization of the structured textual representation into a representation format for further processing. Resources and relations provide the information in the same fashion as the textual representation, i.e., a string-based value assertion, reflecting individual concepts, their axioms, and their relations. Thus, this model reflects the network-like structures of Semantic Web data with additional grouping and classification of triples. Other use cases such as the mappings of SPAQL query results require a-priory knowledge about the structure and how it is mapped to the `Resource-Relation` model. However, the separation of concern paradigm allows for adjusting the `Parser Component` for different data sources and structures. Figure 7.6 illustrates the reorganization and classification of an example definition of an `owl:DatatypeProperty` into a relation.



```
foaf:name rdf:type          owl:DatatypeProperty ;
          rdfs:comment      "A name for some thing." ;
          vs:term_status    "testing" ;
          rdfs:isDefinedBy  foaf: ;
          rdfs:label        "name" ;
          rdfs:subPropertyOf rdfs:label ;
          rdfs:range        rdfs:Literal ;
          rdfs:domain       owl:Thing ;
```

```
Annotations = { "rdfs:comment"     : ["A name for some thing."],
                "rdfs:isDefinedBy": ["foaf:"],
                "rdfs:label"      : ["name"],
                "vs:term_status"  : ["testing"]
              },
Axioms = { "rdfs:subPropertyOf"   : ["rdfs:label"] },
Types = ["owl:DatatypeProperty"]
DomainRangePairs=[ {domain:"owl:Thing", range:"rdfs:Literal" } ]
```

Figure 7.6: Arrows indicate the classification of triple statements into annotations, axioms, types, and domain-range pairs. *Note:* The Resource-Relation model is a simple reorganization of the retrieved textual data into a JSON object. The customizable parser component specifies how different elements are classified.

Mapper Module

This module consists of two components, i.e., the `Vertex-Edge` mapper and the `Node-Link` mapper. Each mapper provides a corresponding data model. In the following, we describe the mappers and the corresponding data models.

Vertex-Edge Mapper: The `Vertex-Edge` mapper is responsible for the transformation of the `Resource-Relation` model into a graph structure using vertices and edges. These mappings define the path specifications for the corresponding data, e.g., the display name is assigned using the path `annotations.rdfs:label`. Additionally, the `Vertex-Edge` mapper creates edges between vertices for resource axioms. These edges have the type `axiomEdge`, and their identifier is derived from the source and target identifier (i.e., `sourceId$$axiom$$targetId`). This design decision reflects the triple structure and accounts for multiple usages of axiom constructs such as *rdfs:subClassOf*. Relational axioms, such as *rdfs:subPropertyOf*, are not mapped in this component. Essentially, relations (or properties) are also “resources”. However, we argue that their incorporation results in a larger graph structure that requires what we call “auxiliary” vertices and edges. Furthermore, these kind of axioms are subject to T-Box definitions and do not appear in A-Box definitions. We argue that manipulating the graph structure by introducing “auxiliary” elements is a task for another mapper component. Therefore, the

`Vertex-Edge Mapper` creates a basic graph structure reflecting resources, i.e., their direct relations and axioms.

Vertex-Edge Model: The `Vertex-Edge` model is designed to reflect a basic graph structure using vertices and edges. Vertices are derived from resources and provide the type, name, and identifier attributes. Edges are derived from relations and provide alongside the type, name, and identifier attributes additionally source and target attributes for the connection between vertices. Vertices and edges provide references to corresponding resources and relations, respectively.

Node-Link Mapper: The `Node-Link` mapper modifies the basic graph structure of the `Vertex-Edge` model using merge, split, and nesting functions. Additionally, the mapper allows for creating auxiliary nodes and links that address domain range restriction and relational axioms. Similar to the `Vertex-Edge` mapper, these mappings define the path specifications for the corresponding data using a definition map. The definition map specifies the criteria when manipulation operations are performed on the graph.

The VOWL notation [82] provides an example of the necessity for merge and split operations. For example, *owl:equivalentClass* axioms result in a merged node for participating elements. Literals are separated into individual nodes for each link. This mapper performs merge and split operations based on the link types. A link consists of a source node and a target node, forming their interrelation. Merge operations extend the source node with a reference to the target node. The target node is then removed from the graph structure, and its links are assigned to the source node. Split operations create a clone of the target node. Corresponding links are modified to use the cloned element as the new target element. The UML notation provides an example of nesting functions. For example, the links of type *owl:DatatypeProperty* and their ranges are assigned to the node and then removed from the graph, allowing for the creation of nested visual representations. Auxiliary nodes allow us to represent "resources" such as *owl:ObjectProperty* and *owl:DatatypeProperty*. The direct link between two nodes is removed and replaced with the auxiliary node, which provides a domain and a range link, respectively. Furthermore, the auxiliary nodes allow us to represent relational axioms such as *rdfs:subPropertyOf* as links between them. Our framework defines criteria for different graph manipulation operations (i.e., merge, split, nesting, and introduction of auxiliary nodes). Other mappings that are not described through our manipulations are subject to the mapper implementations.

Node-Link Model: The `Node-Link` model consists of nodes and links. Nodes have an id, a type, and a name. Links have a source and a target node to form the connections.

Rendering Module

The rendering module performs all visualization-related tasks.

Rendering Component: This component coordinates other visualization components and receives the node-link model as input. Its primary task is to create customizable visual primitives for nodes and links. Their visual appearance and spatial position are assigned in corresponding components. Visual primitives are created with references to their corresponding nodes and links. This enables access to information that is provided by other data models. Optional interactions such as zoom, pan, and drag operations are defined in the interactions component.

Visual Appearance Component: As similar as in GizMO, the visual appearance component provides customization for the definition of visual primitives. Configuration objects define, for example, the geometric shape (e.g., circle, rectangle, and ellipse) and its CSS attributes such as background color and stroke styles. While creating the visual primitives, the rendering component retrieves corresponding configuration objects for nodes and links that are based on their type. Thus, nodes and links with different types are rendered accordingly.

Spatial Layout Component: The different implementations of layout algorithms (e.g., force-directed layout or tree-layout for node-link diagrams) are subject to this component. This component receives as input the visual primitives for nodes and links and assigns their spatial position in the graph.

Interactions Component: The interactions component implements the user interactions. User interactions are categorized into three groups: graph (e.g., zooming and panning), node, and link interactions. Drag, hover, and click interactions are configurable individually for nodes and links. Additionally, the type of nodes and links allows for the configuration of different interactions based on their type.

7.2.2 Visualizing Ontologies

Our first use case addresses the visualization of ontologies. The `Data Access Handler` component uses a system call on the proxy side and processes the ontology using the OWL2VOWL converter[1]. Our design decision is based on the fact that OWL2VOWL uses the OWL-API and exports ontologies in a JSON format. Furthermore, the OWL-API supports various serialization formats and the loading of imported ontologies that

[1]https://github.com/VisualDataWeb/OWL2VOWL

are described using the *owl:imports* statements. OWL2VOWL additionally handles the assignment of missing domain range restrictions for properties to *owl:Thing*. However, this is merely a technical design decision. Other tools such as Apache Jena can be used to process the ontology data. Our design decision allows for reusing the array of existing applications to create a `Data Access Handler` for ontologies that only requires its URI specification. In this use case, the parser component is designed to process the VOWL JSON format and convert it into the `Resource-Relation` model.

The `Vertex-Edge` mapper transforms the `Resource-Relation` model and creates vertices and edges. Vertices and edges have a display name that is derived from the annotation *rdfs:label*. The types are assigned using the semantic types of resources and relations (e.g., *owl:Class*, *owl:DatatypeProperty*, etc.). Edges between vertices are created using the relations and the axioms of resources.

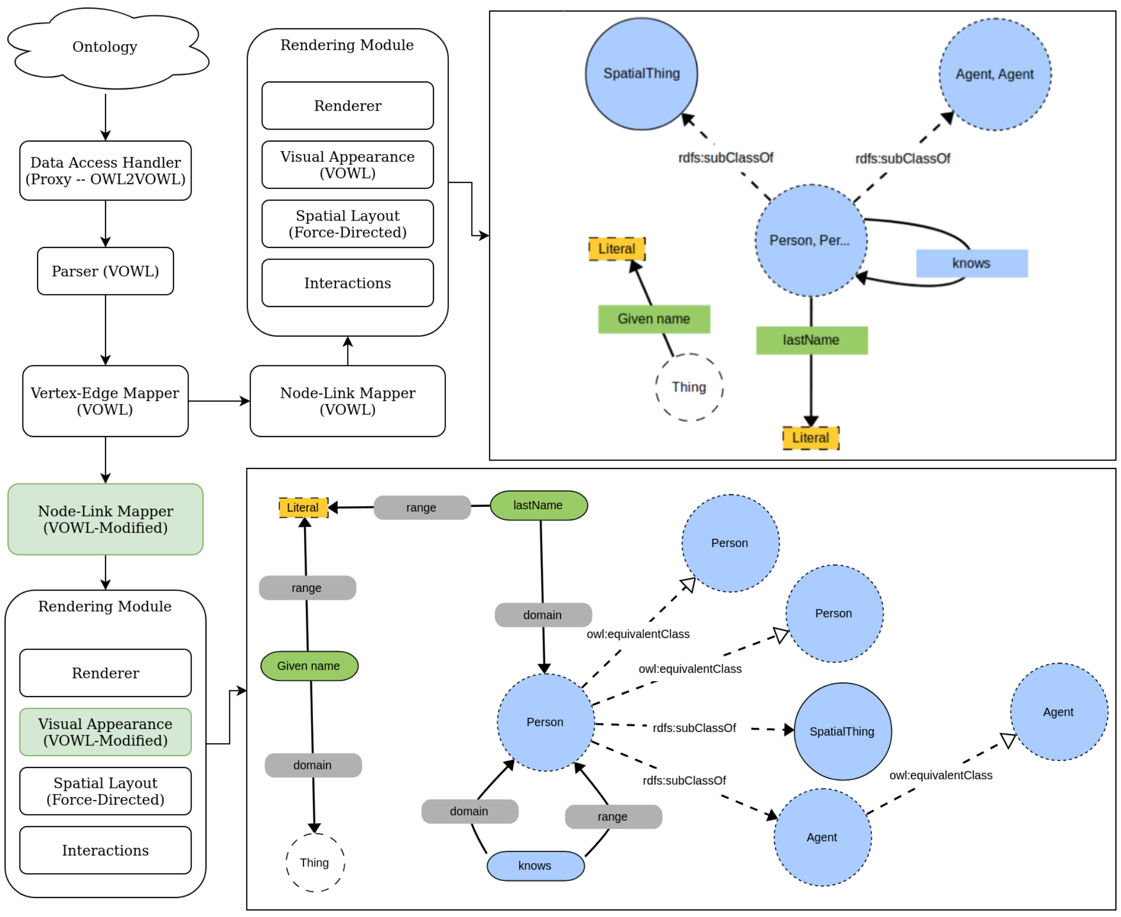


Figure 7.7: A pipeline with branches representing two pipelines for the visualization of the same example ontology. The upper branch shows the visualization using the VOWL notation. The lower branch highlights modified components in light green. Its visualization introduces auxiliary nodes and removes merge and splitting operations.

The `Node-Link` mapper manipulates the graph structure of the `Vertex-Edge` model. These manipulations are subject to the targeted visual notations. In this example, we perform the same merge split operations as described in the VOWL notation, e.g., merge *owl:equivalentClass* axioms and split *rdfs:Literal*s. The visual appearance component defines the configurations for different types according to the VOWL notation. Figure 7.7 illustrates the pipeline and the resulting visualization. Alongside the VOWL representation, we only modify the `Node-Link` mapper and its visual appearance definitions to showcase the visualization of ontologies with auxiliary nodes for domain range restrictions.

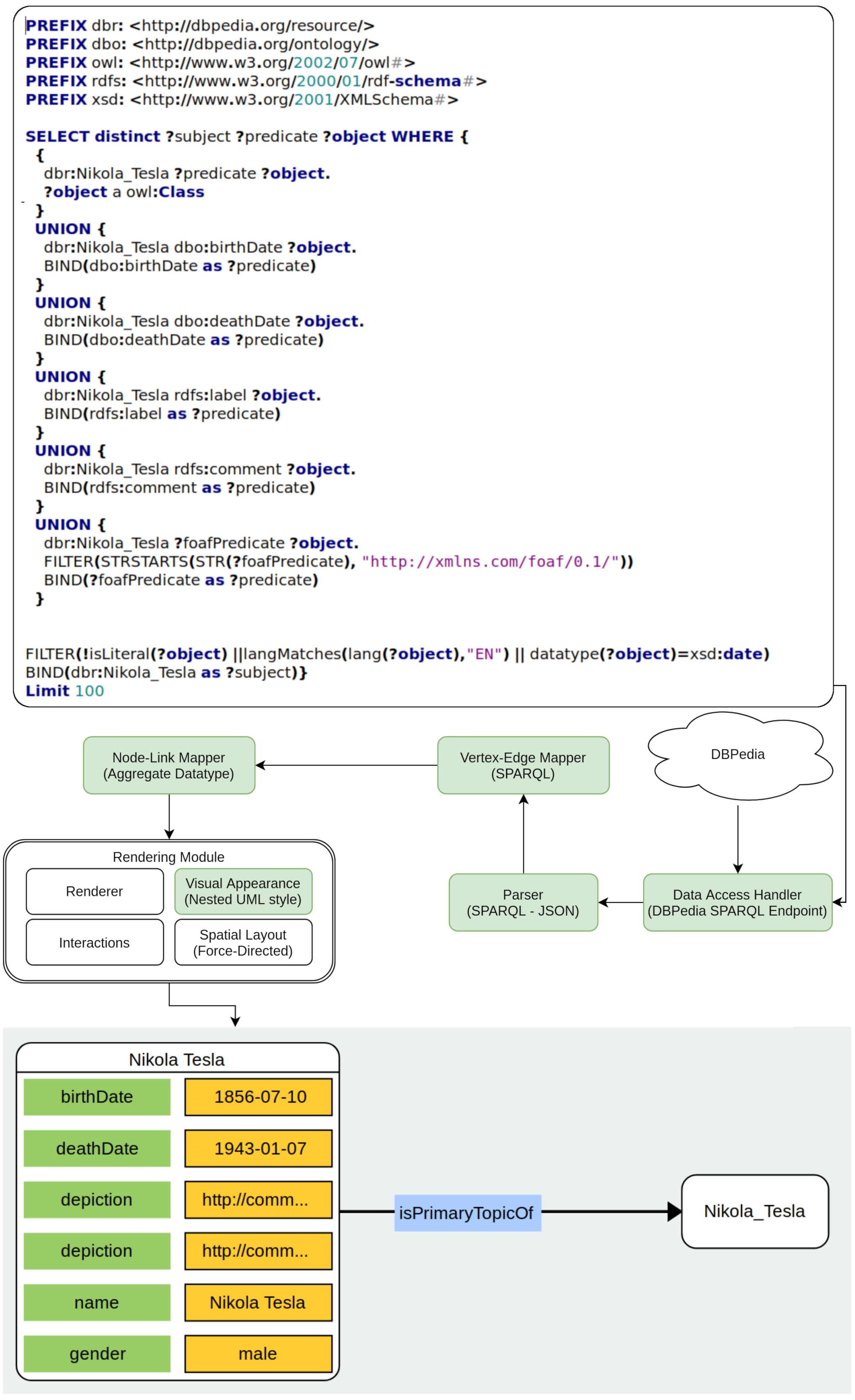
7.2.3 Visualizing SPARQL Query Results

We describe the visualization of the SPARQL query results retrieved from DBpedia as our second use case. The `Data Access Handler` component specifies the URL, graph, query, and a suffix. The suffix specifies the parameters for the HTTP request retrieving the result as JSON. However, we do enforce one condition for the queries. We create only "simple" queries as "?subject ?predicate ?object" triple pattern in the select clause. This reflects the underlying data structure within the endpoint and eases up the implementation of the parser component.

The parser component transforms the results into the `Resource-Relation` model. Subjects are mapped to resources, known predicates, such as *rdfs:label*, are assigned to annotations, and the corresponding object is omitted as a resource. The omitted value is assigned as the value of the annotation. Predicates that can not be assigned to axioms, type assertions, or annotations are treated as relations between two resources.

This use case reuses the already implemented components for the `Vertex-Edge` mapper, the `Node-Link` mapper, and the rendering components. The `Vertex-Edge` mapper is customized to retrieve the display name for links using the resource URI. The `Node-Link` mapper is customized to perform nesting operations on *owl:DatatypeProperty* links. A link-type validation assigns *owl:DatatypeProperty* as the type for links targeting a literal. Furthermore, the display names of links are adjusted to show the suffix of its corresponding URI. These adjustments in the `Node-Link Mapper` account for the missing type and label assertions of predicates for the retrieved SPARQL results. The `Visual Appearance` component is customized to display nested attributes. Figure 7.8 shows the visualization of a simple SPARQL query for DBpedia. The implementation and the discussed examples are available on GitHub.[2]

[2] `https://anonymousninjaturtle.github.io/PipelineVisualizations/`



```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT distinct ?subject ?predicate ?object WHERE {
 {
  dbr:Nikola_Tesla ?predicate ?object.
  ?object a owl:Class
 }
 UNION {
  dbr:Nikola_Tesla dbo:birthDate ?object.
  BIND(dbo:birthDate as ?predicate)
 }
 UNION {
  dbr:Nikola_Tesla dbo:deathDate ?object.
  BIND(dbo:deathDate as ?predicate)
 }
 UNION {
  dbr:Nikola_Tesla rdfs:label ?object.
  BIND(rdfs:label as ?predicate)
 }
 UNION {
  dbr:Nikola_Tesla rdfs:comment ?object.
  BIND(rdfs:comment as ?predicate)
 }
 UNION {
  dbr:Nikola_Tesla ?foafPredicate ?object.
  FILTER(STRSTARTS(STR(?foafPredicate), "http://xmlns.com/foaf/0.1/"))
  BIND(?foafPredicate as ?predicate)
 }


FILTER(!isLiteral(?object) ||langMatches(lang(?object),"EN") || datatype(?object)=xsd:date)
BIND(dbr:Nikola_Tesla as ?subject)}
Limit 100
```

Figure 7.8: A pipeline representing the visualization of SPARQL queries. The modified components in comparison to the first pipeline (i.e., native VOWL representation in Figure 7.7) are highlighted in light green. Its visualization introduces nesting functions for a UML-based representation.

7.2.4 Pipeline Configuration

We realize the visualization generation process using a customizable pipeline-based approach. Our approach uses the separation of concern paradigm to refine the visualization generation process. The refined steps are accompanied by data models and customizable components grouped in modules for the flexible realization of different visualizations for various use cases and user groups. The individual modules, components, and data models are conceptualized as stand-alone artifacts allowing for their modification and extensions for specific use cases and corresponding visualization tasks. However, the orchestration of individual components is required for customizing and arranging them in such a way that they fulfill the requirements of a specific use case. On top of the individual artifacts, we provide a customizable visualization pipeline configuration application. This prototype application serves as a WYSIWYG pipeline builder allowing users to configure and select components for different use cases.

In this application, the pipeline can access four example data sources, demonstrating its flexibility in handling different data sources such as ontology files, SPARQL queries for DBpedia and WikiData, and RESTful-API calls to the Open Research Knowledge Graph. The node-link mappers demonstrate different graph manipulations. The rendering module creates node-link diagram visualizations that are adjustable w.r.t. their visual appearance that is based on GizMO [116]. The rendering module implements basic user interactions such as zooming, dragging and panning. A force-directed layout is used to create the spatial assignment of nodes and links in the graph visualization. Figure 7.9 gives an overview of the UI of the framework.

The limitations of the current prototype implementation are twofold. First, the UI for creating pipelines is fixed with respect to the arrangement of the components, i.e., data source selection, `Vertex-Edge` mapper, `Node-Link` mapper, and rendering module. Second, our prototype implementation addresses only the visual representations in the form of node-link diagrams. Thus, the data models and the pipeline composition are tailor-suited for such visualization types. Furthermore, the components use a top-down communication where their results feed into the next component in the pipeline. We envision the bottom-up communications to enable the creation of applications that allow through means of visual editing also to modify the content provided by data source. For example, changes in the graph propagate backward through the pipeline and create updates for the data source, e.g., SPARQL query updates, RESTful-API calls for creating new entries, or integration with version control systems such as git.

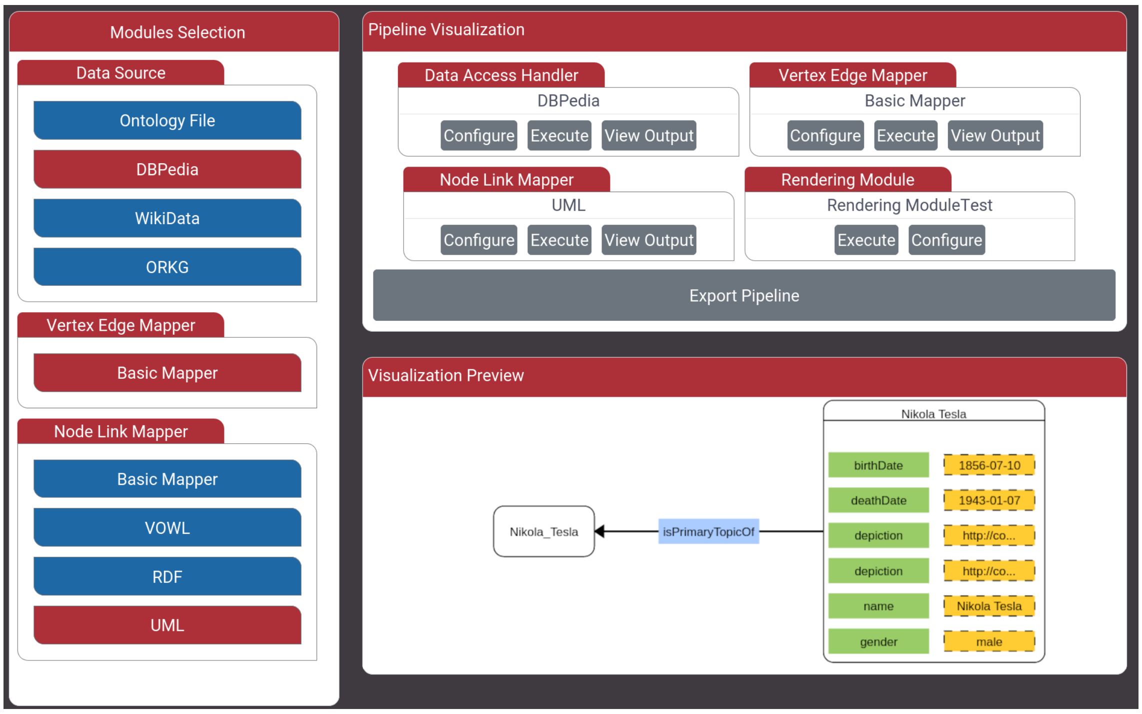


Figure 7.9: Overview of the UI of the framework. Left side: Module selection for data sources, vertex-edge mapper, and node-link mapper. *Note: This version uses a single vertex-edge mapper*. Top: Pipeline configuration. Bottom: Visualization preview for the configuration of components.

Exporting Pipelines

The main objective of this application is to provide the configured pipeline to the user, allowing them to extend and modify the components to the individual targeted use case. Our approach exports the created pipeline as a zip file creating source code for a fully functional React core application that serves as an entry point for development. Separation of concerns for individual components and the pipeline organization allow developers to adjust and customize individual components to their needs and the requirements of the underlying use case. The implementation of our components uses class inheritance. Thus, new components can be derived from existing ones that serve as examples. We demonstrate the application in a video on Youtube[3], showcasing how easily we can create pipelines and export those as React applications. These React applications serve as a basic source code infrastructure for developers.

[3]https://youtu.be/0bGKTkVTQbU

7.3 Discussion and Chapter Summary

This chapter presented a pipeline-based approach for the visualization of Semantic Web data. Our approach applies the separation of concerns paradigm for the commonly used steps in the visualization generation process. We argue that this refinement and a modular pipeline architecture increase the flexibility and foster the creation and reuse of various visual representations. The objective of our approach is to facilitate the creation of visual representations and provide a unified visualization framework for creating visualizations in Semantic Web contexts.

As described in Section 7.1.5, our approach uses divergence in components and convergence in data models. The pipeline-based approach allows for creating *the right components for the right task*. While we address the customizable visual representation in the form of node-link diagrams, the approach itself is not limited to this specific visualization method. We argue that the pipeline-based approach forms the foundation to create various visual representations for different use cases and targeted user groups.

Alongside the approach, we introduce three data models that serve as convergence points within the pipeline. The `Resource-Relation` model provides the reorganization of the input data model into types, annotations, and axioms. Relations additionally provide domains and ranges that reflect the connection between resources. The corresponding implementation of the parser component is accountable for the coverage of the input data model. The `Vertex-Edge` model reflects a simple graph structure. This model results from transforming the `Resource-Relation` model using the customizable `Vertex-Edge` mapper component that selects the relevant information for the visualization. We introduce the `Node-Link` model as the means to manipulate the graph structure for the resulting visualization. These manipulations are defined in the customizable `Node-Link` mapper component. While our data models are created with the purpose of providing convergence points in the pipeline and enable the customizable visualization of node-link diagrams, components for other use cases with other data models can be created for the pipeline.

We provide our implementation in JavaScript. This fosters the creation of new components using class inheritance and also reduces the from-scratch implementation. Different use cases and visualization methods provide their own implementations and customization for the visualization pipeline. Due to the separation of concern paradigm and the stand-alone artifact conceptualization of components, new use cases and visualizations can directly reuse existing components of other pipelines. However, the reuse is dependent on the availability of their implementations. Our current implementation targets to showcase the applicability of the approach for a predefined set of use cases. We argue that the creation of different components for different use cases is a community effort.

The prototype implementation of individual components provides a basic infrastructure to design the data flow that results in a node-link model. This model serves as input for the rendering module that takes additional input from components responsible for visual appearance, spatial layout, and optional interactions within the visualization. The modules and components can be orchestrated and configured in a WYSIWYG application. This application allows exporting the configured pipeline that serves as an entrance point for further development.

The approach and its prototype implementation results provide evidence to answer **RQ3**. Visualization pipelines are flexible and extendable, allowing for facilitating the process of creating visual representations for different use cases and various user groups. Our approach is flexible and extendable through modularization and componentization and allows for creating visual representations for different data sources using the customizable components. Our approach refines the commonly used steps in the visualization generation process using the separation of concern paradigm. The pipeline-based approach allows for creating *the right components for the right task.* Our approach targets a unified visualization framework for Semantic Web data, which we address through divergence of customizable components and convergence in data models. A visual pipeline builder allows to configure a pipeline in a WYSIWYG manner and export it as source code for React applications. The source code for individual components is published under the MIT license. Developers can quickly initialize the source code infrastructure using the visual pipeline builder, adjust and customize components to current use case and user needs, and create pull requests to increase the reusability of the newly created components.

CHAPTER 8

Conclusion

In this thesis, we investigated the challenges of visual exploration of Semantic-Web-Based knowledge structures. After describing the challenges and the research problem in Chapter 1, the following two chapters provide the necessary background concepts (cf. Chapter 2) and an overview of related work (cf. Chapter 3). The four following chapters (cf. Chapter 4, 5, 6, and 7) present the main contributions and proposed solutions for the challenges in the visual exploration of Semantic-Web-Based knowledge structures. In this chapter, we review the stated research questions and examine the achieved results. Additional use cases and opportunities for future work are described in Sections 8.2 and 8.3.

8.1 Analysis of Research Questions

We study the main research problem by separating it into three more specific research questions that are individually examined. Research question **RQ1** examines how to facilitate the creation and modification of Semantic-Web-Based knowledge structures from a user perspective. Knowledge modeling is often done in collaborative efforts, thus, requiring approaches to serve the needs of different user groups with various backgrounds. Research question **RQ2** examines how to facilitate understanding of Semantic-Web-Based knowledge structures. Visualizations exploit humans' cognitive ability to understand complex data through visual representations. However, visual representations are highly dependent on the individual use case and targeted user groups. Research question **RQ3** examines how to ease the creation of visual representations for specific use cases.

Research Question 1 (RQ1)

How can we ease the creation and editing process of Semantic-Web-Based knowledge structures from a user perspective?

In Chapter 4, we answer this question by showing that web-based visual modeling approaches are capable of reducing entrance barriers in the creation and editing processes of Semantic-Web-Based knowledge structures. We present requirements for a device-independent visual modeling approach for ontology development. Device independence is becoming more and more important in ontology modeling for different reasons: Diverse types of computing devices, such as tablets, smartphones, convertibles, and touchscreens, are increasingly used in work environments. Nowadays, knowledge workers often use more than one device for their daily tasks in a multitude of interaction contexts, ranging from classical desktop settings to mobile scenarios in meetings, workshops, and on business trips. Additionally, visual modeling approaches allow for engaging domain experts who are less familiar with semantic formalism and conceptual modeling techniques.

A preliminary user study compared the device-independent approach with two other web-based applications (i.e., WebProtégé and TurtleEditor). The results indicate the additional benefits of the proposed approach, which are reflected in better average scores and faster modeling times (cf. Section 4.1.4). However, this study is considered only preliminary due to the small group of participants. A follow-up user study compared the proposed visual ontology modeling approach, based on node-link diagrams, with a modeling paradigm that uses hierarchical trees and form widgets. In particular, the follow-up study compared two ontology modeling tools: Protégé and WebVOWL Editor, each implementing one of the modeling paradigms. The study indicates that visual ontology modeling, based on node-link diagrams, is comparatively easy to learn and is recommended especially for users with little experience with ontology modeling and its formalizations. We argue that due to the growing attention Semantic Web receives in academic and industrial contexts, the modeling of structured and machine-readable knowledge shifts towards collaborative efforts of knowledge engineers and domain experts. Involving domain experts more directly in ontology modeling requires immediately available approaches that are easy to use and independent of the device and interaction context. The evaluation results provide empirical evidence to answer **RQ1** that device-independent visual modeling approaches facilitate the creation and editing processes of Semantic-Web-Based knowledge structures.

Our approach for device-independent visual modeling of ontologies has the following limitations. It has been designed to reduce entry barriers for users new to ontology modeling who are less familiar with OWL formalizations. Thus, it supports only a subset of OWL constructs that allows for creating small ontologies, including classes, object and datatype properties, class hierarchies using `rdfs:subClassOf`, and some logical constraints such as `owl:disjointWith`. We argue that creating ontologies of sufficient quality requires joint efforts of domain experts and knowledge engineers. The visual modeling approach allows domain experts to draft the ontology, and knowledge engineers create further refinements using ontology modeling tools that provide full OWL formalization features. Thus, our approach provides only the first steps in the modeling of ontologies by engaging different user groups with different backgrounds.

Engaging various user groups in the ontology development process using a device-independent visual modeling approach induces a follow-up research question:

Research Question 2 (RQ2)

How can we improve understanding of Semantic-Web-Based knowledge structures using interactive and user-centered visualizations?

Visualizations serve the purpose of addressing specific information needs for the data at hand and human's ability to understand complex data through visual representations. While various visualization methods and tools exist, suitable visualizations are highly dependent on the use cases and the targeted user groups.

In Chapter 5, we present a methodology and its realization (GizMO) for customizable visualization of ontologies. A customizable visual representation model needs to address the following requirements: **i)** provide the customizable visual appearance of rendering elements in order to coincide with the user's *mental model*; **ii)** provide spatial information and visibility status of rendering elements in order to coincide with the user's *mental map*; **iii)** provide the means to represent and share the definition of visualizations. Our methodology solves these requirements by providing visual definitions in the form of annotation ontologies. Inspired by the Web Annotation Data Model [95], the methodology uses targeting properties to link representational definitions with OWL constructs and individual elements from the visualized ontology. We showcase the applicability of the methodology by providing two applications capable of interpreting the GizMO representation model. The first application targets the creation and editing of visual notations in a WYSIWYG manner. The second application targets the customizable visualization of ontologies allowing for exchanging visual notations on the fly, creating views, and exporting containers for sharing. The central aspect of the methodology is its utilization of OWL for definitions of visual representation models. The methodology separates the visual abstraction into two layers: The *global* layer reflects users' mental model and addresses the customizable visual representation of OWL constructs. The *local* layer addresses the mental map of users and provides the means to customize the spatial arrangement, visibility status, and optional glyph modifications. The methodology facilitates the sense-making of ontologies through its customizable visual representations.

The design decisions for the methodology and the technical realization of GizMO are conceived to facilitate the customizable definitions for the visual representation of ontologies. The success of the methodology and GizMO depends on the integration into other frameworks and tools. GizMO is currently limited in its coverage of OWL constructs and implicit mappings. Some visual representations cannot merely be described by the visual appearance and spatial arrangement of glyphs. For example, nested visualizations such as UML or graph manipulations using node multiplications such as VOWL. Regardless of its limitations, GizMO provides the means to create customizable visual represents for node-link diagrams in the form of annotation ontologies. These annotation ontologies defining visual depiction of ontologies can be reused, adjusted, and shared.

While Chapter 5 addresses the customizable visualization of the underlying structure of Semantic-Web-Based knowledge representations, in Chapter 6, we present an approach towards customizable chart visualizations of tabular data originating from Knowledge Graph. While Knowledge Graphs are often visualized in the form of node-link diagrams to facilitate understanding of its structure, they are not limited to schema or vocabulary data. Chart visualizations provide different views on the data, facilitating its understanding and analysis. However, suitable chart visualizations also depend on the use case and the data at hand. In this approach, we use additional semantics for creating customizable chart representations. A data acquisition process uses a human-in-the-loop approach for transforming tabular data into Knowledge Graph representations and assigns additional semantics to cell values (i.e., metrics and units).

These additional semantics allow us to reconstruct and organize tables in information groups, i.e., sub-tables based on metrics and units. The semantics of *metrics* select suitable visualization from a large space of all chart types. The semantics of *units* perform information organization and align the numerical value representations. Customizations are enabled through chart type selection and axis mappings. Using the paper comparison feature of ORKG [122], the approach realizes advanced use cases, such as the visualization of information distributed among tables in multiple articles and leader-boards.

Our approach builds upon the semantics and the structure of the tabular data representation in a knowledge graph. Thus, it is currently limited to the chosen transformation model. Furthermore, the approach addresses the one-dimensional representation of columns and rows. In our approach, the first column of the table refers to unsorted entries. However, when dealing with order-dependent entries, such as time series or physical distances, the position on the axis (sorting) is significant for information comprehension. Currently, our approach does not address order-dependent entries in the first column.

The current implementation of the approach manages the analysis of the additional semantics using string comparison and ad-hoc rules. The objective of the approach is to enable customizable visualization for tabular data. Thus, we focused on the minimal requirements for the semantics, which we applied to an example table addressing the performance measurements such as *Precision*, *Recall*, and *Runtime*. Regardless of the limitations of our approach, it brings tables of scientific articles to "life" and enables different views on their information.

Suitable visualizations highly depend on the use case, the data, and the targeted user groups. The two approaches and their implementations provide evidence to answer **RQ2**. Approaches with customization capabilities can increase the sense-making of Semantic-Web-Based knowledge structures. Customizations allow users to adjust the visual representations to their current needs.

While having introduced customizable approaches for the visual representations, their implementations target node-link diagrams and chart visualizations. To support different data sources and visualization methods, the following research question emerges:

Research Question 3 (RQ3)

How can we ease the creation of visual representations in Semantic Web contexts for different use cases and diverse audiences?

In Chapter 7, we address the creation of customizable visualization pipelines in Semantic Web contexts. Pipeline-based approaches are often used in applications addressing data visualization. They are flexible and extendable, allowing for the realization of various visualization methods and versatile data sources. The pipeline-based approach allows for creating *the right components for the right task*.

While we focus on the customizable visual representation in the form of node-link diagrams, the approach itself is not limited to this visualization method. Our approach targets a unified visualization framework for Semantic Web data. However, a *one-size-fits-all* solution is challenging, if not impossible, nor feasible, to realize. We address this challenge through the divergence of customizable components and convergence in data models. We identified and refined the commonly used steps in the creation process of visual representations using the separation of concern paradigm. The individual refined steps are accompanied by their respective customizable components. Using the prototype implementation, we showcase achievable pipeline configurations and their visual representation results for different data sources (i.e., ontology files, SPARQL query results for DBpedia and Wikidata, and RESTful API calls for the Open Research Knowledge Graph). The use cases for Wikidata and ORKG are addressed in the visual pipeline builder that allows for creating and customizing pipeline configurations in a WYSIWYG manner. Additionally, the pipeline builder allows for exporting the configured pipelines as React source code, enabling further modifications and customizations. Using class inheritance, new components reuse the existing code base for specific use cases. The approach and its prototype implementation results provide evidence to answer **RQ3**. This project is designed as an open-source community-driven approach, reducing from scratch development and reusing advanced techniques provided by other researchers and developers.

Regarding the main challenge of the *visual exploration of Semantic-Web-Based knowledge structures*, we are able to summarize that customizable and flexible approaches are required for users' needs to facilitate the creation, communication, and understanding of domain knowledge in Semantic Web contexts. Furthermore, pipeline-based approaches facilitate the generation process of visual representations by allowing developers to modify or create new components, therefore reducing *from scratch development*.

8.2 Additional Use Cases

The obtained findings from this thesis have already partially been applied in the graph visualizations of the Open Research Knowledge Graph (ORKG). While the graph visualization implementation builds upon GizMO, customizable visual representations are not yet available for the users. This particular visual representation thrives through its exploration and animation features, which are closely coupled with the implementation. Through the separation of concern and a pipeline-based conceptualization, we showed that our pipeline-based approach is also able to address the ORKG use case.

Chapter 6 introduced an approach towards customizable chart visualizations of tabular data originating from Knowledge Graphs. The “Self-Visualization-Service” of ORKG partially incorporates the findings of the proposed approach. The "Paper and Contribution" comparison feature provides the input table for the chart generation. A human-in-the-loop approach selects the columns of interest and provides a corresponding data type. A user interface allows users to select a single column for the x-axis and multiple columns for the y-axis. Currently, the “Self-Visualization-Service” supports a small set of chart visualization methods. Its implementations focused on the persistent integration of created visualizations with the ORKG backend infrastructure.

The European project “Semantically Coordinated Semiconductor Supply Chains” (SC^3)[1] aims to facilitate the development of ontologies in the semiconductor domain and related supply chain domains. Ontology development tools such as Protégé are typically designed for experts, supporting full OWL formalizations. However, they are challenging to learn for users new to ontology modeling. Furthermore, in industrial contexts, administrative privileges for installing new software on a system pose as hard restrictions, increasing the entrance barrier for users to get engaged in ontology modeling.

This project highlights the need for a user-centered approach that can facilitate communication between different stakeholders and provide a collaborative environment for the creation of *standardized* ontologies in the semiconductor domain. In this project, we will combine the results from all three research questions under one umbrella, i.e., visual modeling, customizable visual representations (GizMO), and a pipeline-based architecture for flexible adoption to users’ needs. Furthermore, in this project, we introduce a hybrid approach combining three modes of operation in order to address various user groups involved in ontology modeling.

As the most crucial requirement for the approach, we identified that it has to serve the users’ needs from various audiences with diverse backgrounds and in different contexts. Web-based approaches reduce entrance barriers and engage different user groups more directly because such applications are ready-to-use without requiring the installation of additional software. We define two general user groups involved in ontology modeling: **a)** non-experts (e.g., domain experts that are new to ontology modeling) and **b)** knowledge engineers who are familiar with OWL notation and its formalizations. We argue that fulfilling the requirements for these two extreme groups will also address the intermediate

[1] SC^3: https://cordis.europa.eu/project/id/101007312

users. We define the requirements for non-expert users as follows: The approach should be **i)** easy-to-learn and easy-to-use, **ii)** provide means to reduce or completely remove the modeling complexity of OWL, **iii)** provide guidance and suggestions for best practices during the modeling. We define the requirements for expert users as follows: The approach should **iv)** support all modeling features of Protégé.

Visual modeling paradigms in the form of node-link diagrams can address the requirements **i)** and **ii)**. However, they have to provide additional customizations for the visual representation to facilitate understanding. The requirement **iii)** can be addressed using auto-complete functionalities to align the created nodes and links with existing terms of ontologies, reducing the manual labor of transforming a high-level conceptualization to OWL elements. Furthermore, guidance towards best practices, such as naming conventions, are addressed by using the auto-complete functionalities. To support all OWL modeling features **iv)**, we envision the use and extension of the OWL-API, which is the backbone of Protégé. Additionally, to engage diverse groups in the modeling process, we envision three modes of operation for modeling: **1)** Textual-Modeling: Expert users may find it faster to create complex OWL axioms such as `owl:Restriction` using textual input fields (e.g., using the Turtle syntax). **2)** Widget-Based-Modeling: For intermediate users who are familiar with Protégé, the learning curve will be minimal since they are already familiar with the usage of the widget-based system. **3)** Visual-Modeling: High-level visual modeling, hiding the modeling complexity for novel users.

We define additional requirements for the realization of a collaborative ontology development framework. The approach should **v)** foster communication between various stakeholders. It has to allow for creating and sharing different views on the data (i.e., visual appearance and spatial layouts), allowing for communication on all abstraction levels involving domain experts and knowledge engineers alike, accompanying Shneiderman's visual information seeking mantra "overview first, zoom and filter, then details-on-demand" [60]. Figure 8.1 illustrates different user cases for various user groups.

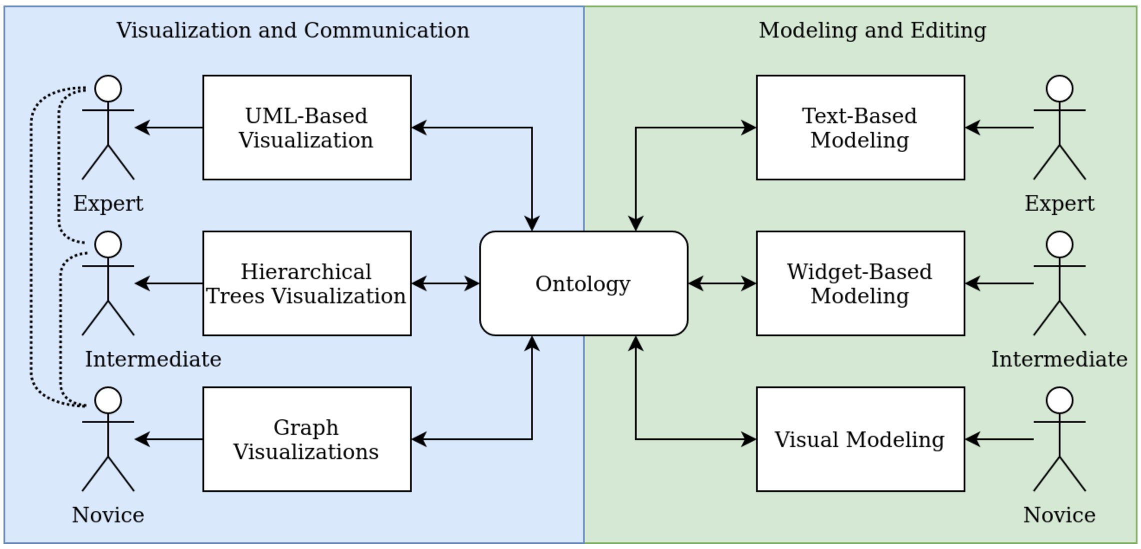


Figure 8.1: Schematic overview for different interactions and user groups.

A fundamental aspect of the Semantic Web is to develop a common understanding and the machine-readable conceptualization of the information and data in certain domains. Thus, the created ontologies and their conceptualizations thrive from discussions and joint agreement upon their definitions and their usage. Thus, we propose a two-fold solution approach. First, similarly as used in version control systems such as git, editing of concepts results in suggestions with “commit” messages. The curating authorities can integrate the changes in the same fashion as “pull or merge” requests with “cherry-picking”, increasing the trust in the created ontology through human validation. Second, Blockchain technologies enable us to create the versioning and history of changes. Thus, integrating Blockchain technologies with ontology development will foster trust and validation of ontologies and enable long-term support for downward compatible systems using old versions of an ontology.

Through the use of a modular conceptualization and standalone component architecture, the obtained results (i.e., software products) are reusable for other related projects. For example, the NFDI consortium (Deutsche Forschungsgemeinschaft – DFG) addresses the development and provenance of ontologies in different domains. While the data storage, provenance, user roles, and access rights may be realized using different technologies, the user interface requirements, providing an entry point for the ontology data, remain similar. Thus, our customizable user-centered approach is applicable across different projects too. Although requirements are a-priory not known or can change over time, the pipeline-based approach already provides a conceptualization and foundation for further development.

8.3 Future Work

In this thesis, we addressed the challenges in the visual exploration of Semantic-Web-Based knowledge structures to facilitate the interaction with and understanding of Semantic Web data. We proposed customizable and user-centered applications to address different use cases, data sources, and targeted user groups. This section addresses future work for the individual approaches presented in this thesis.

In Chapter 4, we presented an approach for device-independent visual modeling by considering the different input/output modalities and brought them into synergy for different devices and display sizes. As this approach is designed to engage user groups less familiar with ontology modeling techniques, it provides only a subset of OWL constructs. Future work will address the extension towards all OWL constructs and additional guidance for domain experts for visual modeling.

In the preliminary user study, we observed that sketches (using pen and paper) are used to organize the thoughts about the conceptualization. Thus, additional future work investigates how to transform sketches (on paper or a device) to facilitate the initial drafting of an ontology. First steps into this direction have been made by OntoSketch [101], where the users can model ontologies on a tablet using touch-pen interactions. Most touch devices have a camera and internet access, thus by means of computer vision, sketches on paper can be analyzed and transformed towards a simple ontology model. However, the assignment to individual OWL constructs remains a human-in-the-loop approach due to the aspect that users employ different visual notations for sketching (e.g., circles for classes, arrows for links, and rectangles for datatypes).

Our approach focuses on the synergy between touch and pointer (mouse) operations. However, modern devices have additional input modalities, such as acceleration sensors or cameras, which provide opportunities for further interactions with Semantic-Web-Based knowledge structures and their creation and editing. Thus, future work will investigate different input modalities for various device types.

In Chapter 5, we introduced a methodology for the definition of customizable visual representations of ontologies using annotation ontologies. We realized the methodology in the form of a Graph Visualization Meta Ontology (GizMO) for customizable node-link diagram visualizations of ontologies. While GizMO uses ad-hoc rules to provide nested node visualizations, we presented how to overcome its limitations in Chapter 7, where the graph data model of an ontology is processed in three individual mappers. These mappers provide the selection of elements for the visualization and create a simplified graph structure (i.e., `Vertex-Edge` model) which is modified in the `Node-Link` mapper, realizing graph manipulation operations for various visual notations.

Future work will address the realization of different visualization methods (e.g., Euler-diagrams, chord diagrams, and treemaps) using the proposed methodology. Future work will investigate additional transformations between different visualization methods. Different views on the data can support the sense-making of the underlying information and conceptualization presented in ontologies. Thus, enabling transformations between visualization methods will provide further customizations and engage more users.

In Chapter 6, we examined minimal requirements for customizable chart visualizations of tabular data originating from Knowledge Graphs. While node-link diagrams are often used to display the structure of Semantic-Web-Based knowledge structures, with growing size and complexity, these can quickly result in overcrowded visual representations. Chart visualizations can provide a more suitable visual representation of tabular data originating from Knowledge Graphs. Our approach employs a human-in-the-loop technique to transform tabular data into Knowledge Graph representations with additional semantics. These additional semantics serve as the foundation for obtaining views of the Knowledge Graph that feed into various data visualization. Using the additional semantics, our approach recreates tables from Knowledge Graphs and enables the analysis of their content for the creation of customizable chart visualizations.

Future work will address the extension for the definition of additional semantics related to order-dependent entries for the first column. The semantics of *Metrics* define the interplay among them and which chart visualizations are suitable. Thus, future work will address the many definitions of metrics. Additionally, we plan to investigate the alignment to existing vocabularies related to units [124] and the RDF Data Cube Vocabulary [125] in order to increase the flexibility and robustness of the approach. Furthermore, we argue that pattern matching and sub-graph identification will enable the realization of semi-automated generation for context items that guide the information organization and the analysis, enabling the chart visualization of non-tabular data from Knowledge Graphs. A preliminary assumption of the approach is its a-priory known data transformation model, which allows for retrieving information related to the visual representation. Thus, future work will additionally investigate how to obtain views on data that is not organized in a table, enabling us to apply the customizable chart visualization approach to different Knowledge Graphs with different structures.

In Chapter 7, we present an approach that refines the commonly used steps in the visualization generation process. Our approach uses a customizable pipeline-based realization and a modular architecture to increase the flexibility in creating various visual representations. While our approaches focus on the realization of customizable pipelines for node-link diagram visualizations, the general pipeline-based architecture allows for creating different mappers and models using class inheritance conceptualizations. Thus, future work will address the extension of our refined approach towards other visualization methods accompanied by corresponding data models and mappers.

Furthermore, we plan to extend components to enable bottom-up communication between them. Currently, components use a top-down communication where their results feed into the next component in the pipeline. We envision the bottom-up communications to enable the creation of applications that allow through means of visual editing also to modify the data source. For example, changes in the graph propagate backward through the pipeline and create updates for the data source, e.g., SPARQL query updates, RESTful-API calls for creating new data, or integration with version control systems such as git.

Closing Remarks

In conclusion, we hope that the proposed solutions for the individual research questions will foster the engagement of different user groups in Semantic Web contexts and are useful to developers, researchers, ontology engineers, and domain experts.

“If you want to find the secrets of the universe, think in terms of energy, frequency, and vibration.” – **Nikola Tesla**

Bibliography

[1] G. A. Miller, *Informavores. The study of information: Interdisciplinary messages*, 1983 (cit. on p. 1).

[2] J. M. Carroll, *HCI models, theories, and frameworks: Toward a multidisciplinary science*, Elsevier, 2003 (cit. on p. 1).

[3] A. Singhal, *Introducing the Knowledge Graph: Things, not Strings*, Official Google Blog **5** (2012), URL: https://blog.google/products/search/introducing-knowledge-graph-things-not/ (cit. on pp. 1, 23, 25, 35).

[4] F. van Harmelen and D. McGuinness, *OWL Web Ontology Language Overview*, W3C Recommendation, W3C, 2004, URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/ (cit. on pp. 2, 14, 77).

[5] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne et al., *The FAIR Guiding Principles for scientific data management and stewardship*, Scientific data **3**.1 (2016) 1 (cit. on p. 2).

[6] *FAIR Principles*, URL: https://www.go-fair.org/fair-principles/ (cit. on p. 2).

[7] *RDF 1.1 Concepts and Abstract Syntax*, URL: https://www.w3.org/TR/rdf11-concepts/ (cit. on pp. 2, 16).

[8] *RDF Schema 1.1*, URL: https://www.w3.org/TR/rdf-schema/ (cit. on pp. 2, 17, 18, 114).

[9] *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*, URL: https://www.w3.org/TR/owl2-syntax/#Symmetric%5C_Object%5C_Properties (cit. on pp. 2, 20).

[10] S. Harris, A. Seaborne and E. Prud'hommeaux, *SPARQL 1.1 query language*, W3C recommendation (2013), URL: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (cit. on pp. 2, 21).

[11] *The Gremlin Graph Traversal Machine and Language*, https://tinkerpop.apache.org/gremlin.html, Online accessed on 2021-06-29 (cit. on p. 2).

[12] *GraphQL*, Online accessed on 2021-06-21, URL: https://graphql.org/ (cit. on p. 2).

[13] *Cypher Query Language*, Online accessed on 2021-06-21, URL: https://neo4j.com/developer/cypher/ (cit. on p. 2).

[14] J. Hey, *The data, information, knowledge, wisdom chain: the metaphorical link*, Intergovernmental Oceanographic Commission **26** (2004) 1 (cit. on p. 3).

[15] V. Dimitrova, R. Denaux, G. Hart, C. Dolbear, I. Holt and A. G. Cohn, "Involving Domain Experts in Authoring OWL Ontologies", *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, 2008 1 (cit. on pp. 4, 45).

[16] C. Chen, *Visualizing the Semantic Web: XML-Based Internet and Information Visualization*, Springer, 2002 (cit. on pp. 4, 36, 46, 74).

[17] M. Dudáš, S. Lohmann, V. Svátek and D. Pavlov, *Ontology visualization methods and tools: a survey of the state of the art*, Knowledge Eng. Review **33** (2018) (cit. on pp. 4, 7, 37, 38, 46, 75, 109, 111).

[18] J. R. Wilson and A. Rutherford, *Mental models: Theory and application in human factors*, Human Factors **31**.6 (1989) 617 (cit. on pp. 4, 33, 39, 75).

[19] *W3C Semantic Web Logos and Policies*, https://www.w3.org/2007/10/sw-logos.html (cit. on p. 5).

[20] T. Berners-Lee, J. Hendler and O. Lassila, *The semantic web*, Scientific american **284**.5 (2001) 28 (cit. on p. 13).

[21] *W3C: Extensible Markup Language (XML) 1.0*, URL: http://www.w3.org/TR/REC-xml (cit. on p. 14).

[22] B. McBride, "The resource description framework (RDF) and its vocabulary description language RDFS", *Handbook on ontologies*, Springer, 2004 51 (cit. on p. 14).

[23] W. Abramowicz, S. Auer and T. Heath, *Linked data in business*, 2016 (cit. on p. 15).

[24] T. Berners-Lee, R. Fielding and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*, http://www.ietf.org/rfc/rfc3986.txt, 2005 (cit. on p. 16).

[25] *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*, URL: https://www.w3.org/TR/xmlschema11-1/ (cit. on p. 16).

[26] A. Phillips and M. Davis, *Tags for identifying languages*, tech. rep., 2009, URL: http://tools.ietf.org/html/bcp47 (cit. on p. 16).

[27] *RDF 1.1 Concepts and Abstract Syntax*, URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (cit. on p. 16).

[28] M. Arenas, C. Gutiérrez and J. Pérez, "Foundations of RDF Databases", *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School, Brixen-Bressanone, Italy, Tutorial Lectures*, 2009 158 (cit. on p. 18).

[29] M. K. Smith, C. Welty and D. L. McGuinness, *OWL Web Ontology Language Guide*, W3C Recommendation, World Wide Web Consortium (W3C), 2004, URL: http://www.w3.org/TR/owl-guide/ (cit. on p. 19).

[30] T. R. Gruber, *Toward principles for the design of ontologies used for knowledge sharing?*, Int. J. Hum.-Comput. Stud. **43**.5-6 (1995) 907 (cit. on p. 19).

[31] T. Berners-Lee, *Linked Data - Design Issues*,
URL: https://www.w3.org/DesignIssues/LinkedData.html
(cit. on p. 21).

[32] A. Abele, J. P. McCrae, P. Buitelaar, A. Jentzsch, R. Cyganiak, V. Andryushechkin, J. Debattista and J. Nasir, *Linking Open Data cloud diagram 2021*,
"Online accessed on 2021-06-30", URL: http://lod-cloud.net/
(cit. on p. 21).

[33] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer and C. Bizer,
DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia, Semantic Web **6**.2 (2015) 167 (cit. on pp. 21, 23).

[34] D. Vrandecic and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, Commun. ACM **57**.10 (2014) 78 (cit. on pp. 21, 23).

[35] J. Pérez, M. Arenas and C. Gutiérrez, *Semantics and complexity of SPARQL*, ACM Trans. Database Syst. **34**.3 (2009) 16:1 (cit. on p. 22).

[36] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres et al., *Knowledge graphs*, arXiv preprint arXiv:2003.02320 (2020) (cit. on p. 22).

[37] F. Mahdisoltani, J. Biega and F. M. Suchanek,
"YAGO3: A Knowledge Base from Multilingual Wikipedias",
CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings, 2015 (cit. on p. 23).

[38] L. Ehrlinger and W. Wöß, *Towards a Definition of Knowledge Graphs.*, SEMANTiCS (Posters, Demos, SuCCESS) **48** (2016) (cit. on pp. 23, 24).

[39] H. Paulheim,
Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic web **8**.3 (2017) 489 (cit. on p. 23).

[40] S. K. Mohamed, A. Nounu and V. Nováček,
Biological applications of knowledge graph embedding models,
Briefings in Bioinformatics **22**.2 (2020) 1679, ISSN: 1477-4054,
URL: https://doi.org/10.1093/bib/bbaa012 (cit. on p. 23).

[41] M. Galkin, S. Auer and S. Scerri,
"Enterprise knowledge graphs: a backbone of linked enterprise data",
2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI),
IEEE, 2016 497 (cit. on p. 23).

[42] S. Auer, V. Kovtun, M. Prinz, A. Kasprzik, M. Stocker and M. E. Vidal,
"Towards a knowledge graph for science", *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, 2018 1
(cit. on pp. 23, 94).

[43] S. Preator, *New DBpedia Release – 2016-10*, 2017, URL: `https://www.dbpedia.org/blog/new-dbpedia-release-2016-10/`
(cit. on p. 24).

[44] *Wikidata:Statistics*, Online accessed on 2021-06-30,
URL: `https://www.wikidata.org/wiki/Wikidata:Statistics`
(cit. on p. 25).

[45] *Wikidata Query Service – Number of triples extraction*, `https://query.wikidata.org/#SELECT(count(*)as?num)WHERE{?s?p?o}`,
Number of triples extracted using SPARQL Query. Online accessed on 2021-06-30.
(cit. on p. 25).

[46] T. P. Tanon, G. Weikum and F. Suchanek,
"Yago 4: A reason-able knowledge base", *European Semantic Web Conference*,
Springer, 2020 583 (cit. on p. 25).

[47] C. Newton, *Google's Knowledge Graph tripled in size in seven months, 2012*,
Online accessed on 2021-05-19,
URL: `https://www.cnet.com/news/googles-knowledge-graph-tripled-in-size-in-seven-months/` (cit. on p. 25).

[48] A. M. N. Allam and M. H. Haggag, *The question answering systems: A survey*,
International Journal of Research and Reviews in Information Sciences (IJRRIS)
2.3 (2012) (cit. on p. 25).

[49] B. H. Korte and J. Vygen, *Combinatorial optimization*, vol. 1, Springer, 2011
(cit. on pp. 26–30).

[50] *RDF Semantics*, URL: `https://www.w3.org/TR/rdf-mt/` (cit. on p. 30).

[51] J. Hayes and C. Gutierrez, "Bipartite graphs as intermediate model for RDF",
International Semantic Web Conference, Springer, 2004 47 (cit. on p. 30).

[52] M. Friendly, “A brief history of data visualization”, *Handbook of data visualization*, Springer, 2008 15 (cit. on p. 31).

[53] S. K. Card, J. D. Mackinlay and B. Shneiderman, *Using vision to think*, Readings in information visualization: using vision to think (1999) 579 (cit. on p. 31).

[54] C.-h. Chen, W. K. Härdle and A. Unwin, *Handbook of data visualization*, Springer Science & Business Media, 2007 (cit. on p. 31).

[55] N. Cowan, “The Magical Mystery Four: How Is WorkingMemory Capacity Limited, and Why?”, vol. 19, 1, 2010 51 (cit. on p. 31).

[56] C. G. Healey, K. S. Booth and J. T. Enns, *High-Speed Visual Estimation Using Preattentive Processing*, ACM Trans. Comput.-Hum. Interact. **3**.2 (1996) 107 (cit. on pp. 32, 33).

[57] A. Treisman, *Preattentive processing in vision*, Computer vision, graphics, and image processing **31**.2 (1985) 156 (cit. on pp. 32, 33).

[58] W. B. Rouse and N. M. Morris, *On looking into the black box: Prospects and limits in the search for mental models.*, Psychological bulletin **100**.3 (1986) 349 (cit. on p. 33).

[59] D. A. Norman, *Some observations on mental models*, Mental models **7**.112 (1983) 7 (cit. on p. 33).

[60] B. Shneiderman, “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”, *Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, September 3-6, 1996*, 1996 336 (cit. on pp. 34, 95, 138).

[61] *RDF 1.1 Turtle*, URL: https://www.w3.org/TR/2014/REC-turtle-20140225/ (cit. on p. 35).

[62] M. Horridge, N. Drummond, J. Goodwin, A. L. Rector, R. Stevens and H. Wang, “The Manchester OWL syntax.”, *OWLed*, vol. 216, 2006 (cit. on p. 35).

[63] *NeonToolkit*, URL: http://neon-toolkit.org/wiki/Main_Page.html (cit. on p. 35).

[64] TopQuadrant, *TopBraid Composer(TM)*, https://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/ (cit. on p. 35).

[65] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau and J. Hendler, *Swoop: A web ontology editing browser*, Journal of Web Semantics **4**.2 (2006) 144 (cit. on p. 35).

[66] R. Liepins, K. Cerans and A. Sprogis, *Visualizing and Editing Ontology Fragments with OWLGrEd.*, I-SEMANTICS (Posters & Demos) **932** (2012) 22 (cit. on pp. 35, 38, 51).

[67] B. Vu, J. Pujara and C. A. Knoblock, "D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF", *Proceedings of the 10th International Conference on Knowledge Capture*, 2019 189 (cit. on pp. 36, 43, 93).

[68] A. Langegger and W. Wöß, "XLWrap–querying and integrating arbitrary spreadsheets with SPARQL", *International Semantic Web Conference*, Springer, 2009 359 (cit. on p. 36).

[69] S. Das, S. Sundara and R. Cyganiak, *R2RML: RDB to RDF Mapping Language*, https://www.w3.org/TR/r2rml/, 2012 (cit. on p. 36).

[70] V. Wiens, S. Lohmann and S. Auer, "Semantic Zooming for Ontology Graph Visualizations", *Proceedings of the Knowledge Capture Conference, K-CAP 2017*, ACM, 2017 4:1 (cit. on pp. 36, 39, 60, 87).

[71] V. Link, S. Lohmann, E. Marbach, S. Negru and V. Wiens, *WebVOWL: Web-based Visualization of Ontologies*, Online accessed on 2021-07-01, URL: http://vowl.visualdataweb.org/webvowl.html (cit. on p. 37).

[72] S. Falconer, *OntoGraf*, Online accessed on 2021-07-01, URL: https://protegewiki.stanford.edu/wiki/OntoGraf (cit. on p. 37).

[73] S. Negru and S. Lohmann,
VOWL: Visual Notation for OWL Ontologies (Specification of Version 1.0),
Online accessed on 2021-07-01,
URL: `http://vowl.visualdataweb.org/v1/` (cit. on p. 37).

[74] G. Pal, K. Atkinson and G. Li, "Managing Heterogeneous Data on a Big Data Platform: A Multi-criteria Decision Making Model for Data-Intensive Science", *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2020 229 (cit. on p. 37).

[75] J. Sequeda, *Gra.fo, a visual, collaborative, real-time ontology and knowledge graph schema editor*, Online accessed on 2021-07-01,
URL: `http://www.juansequeda.com/blog/2018/10/19/gra-fo-a-visual-collaborative-real-time-ontology-and-knowledge-graph-schema-editor/` (cit. on p. 37).

[76] R. Liepins, K. Cerans and A. Sprogis,
Visualizing and Editing Ontology Fragments with OWLGrEd.,
I-SEMANTICS (Posters & Demos) **932** (2012) 22 (cit. on p. 37).

[77] H. Zhao and L. Lu,
"Variational circular treemaps for interactive visualization of hierarchical data",
2015 IEEE Pacific Visualization Symposium (PacificVis), 2015 81 (cit. on p. 37).

[78] M. Dudáš, O. Zamazal and V. Svátek,
"Roadmapping and Navigating in the Ontology Visualization Landscape",
Knowledge Engineering and Knowledge Management,
ed. by K. Janowicz, S. Schlobach, P. Lambrix and E. Hyvönen, vol. 8876, LNAI, Springer, 2014 137 (cit. on pp. 37, 46).

[79] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis and E. Giannopoulou,
Ontology visualization methods – A survey, ACM Computer Surveys **39**.4 (2007) (cit. on pp. 37, 46).

[80] M. Lanzenberger, J. Sampson and M. Rester, "Visualization in Ontology Tools", *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '09)*, IEEE, 2009 705 (cit. on pp. 37, 46).

[81] 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", *Proceedings of the 2nd Conference on Creativity in Intelligent Technologies and Data Science*, 2017 133 (cit. on p. 37).

[82] S. Lohmann, S. Negru, F. Haag and T. Ertl, *Visualizing Ontologies with VOWL*, Semantic Web **7**.4 (2016) 399 (cit. on pp. 37, 46, 47, 49, 61, 75, 113, 121).

[83] R. Falco, A. Gangemi, S. Peroni, D. M. Shotton and F. Vitali, "Modelling OWL Ontologies with Graffoo", *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, 2014 320 (cit. on pp. 37, 60, 75).

[84] S. Cranefield and M. K. Purvis, "UML as an Ontology Modelling Language", *Intelligent Information Integration*, vol. 23, CEUR Workshop Proceedings, CEUR-WS.org, 1999, URL: http://ceur-ws.org/Vol-23/ (cit. on pp. 37, 75).

[85] H. Knublauch, *Graphical Ontology Editing with TopBraid Composer's Diagram Tab*, https://www.topquadrant.com/2012/06/29/graphical-ontology-editing-with-topbraid-composers-diagram-tab/ (cit. on pp. 38, 60, 75).

[86] S. Negru, F. Haag and S. Lohmann, "Towards a unified visual notation for OWL ontologies: insights from a comparative user study", *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, 2013 73 (cit. on pp. 38, 75).

[87] P. Haase, S. Brockmans, R. Palma, J. Euzenat and M. d'Aquin, *The NeOn UML Profile for Networked Ontologies*, http://neon-project.org/deliverables/WP1/NeOn_2007_D1.1.2.pdf (cit. on pp. 38, 75).

[88] N. Petersen, A. Similea, C. Lange and S. Lohmann, *TurtleEditor: A Web-Based RDF Editor to Support Distributed Ontology Development on Repository Hosting Platforms*, Int. J. Sem. Comp. **11**.3 (2017) 311 (cit. on pp. 38, 47, 49, 60).

[89] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann and T. Stegemann, "RelFinder: Revealing relationships in RDF knowledge bases", *International Conference on Semantic and Digital Media Technologies*, Springer, 2009 182 (cit. on p. 38).

[90] Neo4j, *Neo4j Graph Visualization*, https://neo4j.com/developer/graph-visualization/, accessed March 2020 (cit. on p. 38).

[91] E. Pietriga, C. Bizer, D. R. Karger and R. Lee, "Fresnel: A Browser-Independent Presentation Vocabulary for RDF", *5th International Semantic Web Conference, ISWC 2006*, vol. 4273, LNCS, Springer, 2006 158 (cit. on p. 42).

[92] E. Pietriga, *IsaViz: A visual authoring tool for RDF*, http://www.w3.org/2001/11/IsaViz, 2003 (cit. on p. 42).

[93] E. Pietriga, "Semantic Web Data Visualization with Graph Style Sheets", *Proceedings of the 2006 ACM Symposium on Software Visualization*, SoftVis '06, Brighton, United Kingdom: ACM, 2006 177, ISBN: 1-59593-464-2, URL: http://doi.acm.org/10.1145/1148493.1148532 (cit. on p. 42).

[94] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski and T. Ideker, *Cytoscape: a software environment for integrated models of biomolecular interaction networks*, Genome research **13**.11 (2003) 2498 (cit. on pp. 42, 76).

[95] B. Young, R. Sanderson and P. Ciccarese, *Web Annotation Data Model*, W3C Recommendation, Online accessed on 2021-05-19: W3C, 2017 (cit. on pp. 43, 78, 79, 133).

[96] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana and M.-E. Vidal, "SDM-RDFizer: An RML interpreter for the efficient creation of rdf knowledge graphs", *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020 3039 (cit. on p. 43).

[97] H. Xiao, M. Huang and X. Zhu, *From one point to a manifold: Knowledge graph embedding for precise link prediction*, arXiv preprint arXiv:1512.04792 (2015) (cit. on p. 43).

[98] V. Wiens, S. Lohmann and S. Auer, "WebVOWL Editor: Device-Independent Visual Ontology Modeling.", *International Semantic Web Conference (P&D/Industry/BlueSky)*, 2018 (cit. on p. 46).

[99] M. R. A. Asmat, V. Wiens and S. Lohmann, "A Comparative User Evaluation on Visual Ontology Modeling Using Node-Link Diagrams.", *ISWC (Best Workshop Papers)*, 2018 1 (cit. on p. 46).

[100] T. Tudorache, *WebProtégé*, https://webprotege.stanford.edu/, 2013 (cit. on pp. 47, 49).

[101] M. Brade, F. Schneider, A. Salmen and R. Groh, "OntoSketch: Towards Digital Sketching As a Tool for Creating and Extending Ontologies for Non-Experts", *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, i-Know '13, ACM, 2013 9:1 (cit. on pp. 47, 140).

[102] N. Petersen, L. Halilaj, I. Grangel-González, S. Lohmann, C. Lange and S. Auer, "Realizing an RDF-Based Information Model for a Manufacturing Company - A Case Study", *ISWC*, 2017, URL: https://doi.org/10.1007/978-3-319-68204-4_31 (cit. on p. 47).

[103] S. Falconer, *OntoGraf*, http://protegewiki.stanford.edu/wiki/OntoGraf, 2010 (cit. on pp. 51, 60).

[104] M. Horridge, *OWLViz*, http://protegewiki.stanford.edu/wiki/OWLViz, 2010 (cit. on p. 51).

[105] B. Bach, E. Pietriga, I. Liccardi and G. Legostaev, "OntoTrix: A hybrid visualization for populated ontologies", *Proceedings of the 20th International Conference on World Wide Web (WWW '11), Companion Volume*, ACM, 2011 177 (cit. on p. 51).

[106] J. J. van Wijk and W. A. A. Nuij, "Smooth and efficient zooming and panning", *9th IEEE Symposium on Information Visualization (InfoVis 2003), 20-21 October 2003, Seattle, WA, USA*, 2003 15 (cit. on p. 54).

[107] Stanford, *Protégé*, https://protege.stanford.edu/ (cit. on p. 60).

[108] J. F. G. Navarro, F. J. Garcı´ a-Peñalvo, R. Therón and P. O. de Pablos, *Usability Evaluation of a Visual Modelling Tool for OWL Ontologies*, J. UCS **17**.9 (2011) 1299 (cit. on p. 61).

[109] A. Katifori, E. Torou, C. Halatsis, G. Lepouras and C. Vassilakis, "A Comparative Study of Four Ontology Visualization Techniques in Protege: Experiment Setup and Preliminary Results", *10th International Conference on Information Visualisation, IV 2006, 5-7 July 2006, London, UK*, 2006 417 (cit. on p. 61).

[110] B. Fu, N. F. Noy and M. D. Storey, "Indented Tree or Graph? A Usability Study of Ontology Visualization Techniques in the Context of Class Mapping Evaluation", *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, 2013 117 (cit. on p. 61).

[111] B. Fu, N. F. Noy and M. D. Storey, *Eye tracking the user experience - An evaluation of ontology visualization techniques*, Semantic Web **8**.1 (2017) 23 (cit. on p. 62).

[112] *MEMORY RECALL/RETRIEVAL*, http://www.human-memory.net/processes_recall.html, Online accessed on 2018-07-17 (cit. on p. 65).

[113] A. Assila, K. M. de Oliveira and H. Ezzedine, *Standardized usability questionnaires: Features and quality focus*, electronic Journal of Computer Science and Information Technology **6**.1 (2016) (cit. on p. 66).

[114] D. L. Scapin and J. M. C. Bastien, *Ergonomic criteria for evaluating the ergonomic quality of interactive systems*, Behaviour & IT **16**.4-5 (1997) 220 (cit. on p. 66).

[115] *Quantitative Studies: How Many Users to Test?*, https://www.nngroup.com/articles/quantitative-studies-how-many-users/, Online accessed on 2018-07-17 (cit. on p. 71).

[116] V. Wiens, S. Lohmann and S. Auer, "GizMO–A Customizable Representation Model for Graph-Based Visualizations of Ontologies", *Proceedings of the 10th International Conference on Knowledge Capture*, 2019 163 (cit. on pp. 74, 102, 115, 126, 165).

[117] N. Cowan, *The magical number 4 in short-term memory: A reconsideration of mental storage capacity*, Behavioral and Brain Sciences **24** (2000) 87 (cit. on p. 75).

[118] V. Wiens, M. Stocker and S. Auer, "Towards Customizable Chart Visualizations of Tabular Data Using Knowledge Graphs", *International Conference on Asian Digital Libraries*, Springer, Cham, 2020 71 (cit. on p. 94).

[119] R. Johnson, A. Watkinson and M. Mabe, *The STM report*, An overview of scientific and scholarly publishing. 5th edition October (2018) (cit. on p. 94).

[120] B. Mons, *Which gene did you mean?*, BMC Bioinform. **6** (2005) 142 (cit. on p. 94).

[121] M. Y. Jaradeh, A. Oelen, K. E. Farfar, M. Prinz, J. D'Souza, G. Kismihók, M. Stocker and S. Auer, "Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge", *Proceedings of the 10th International Conference on Knowledge Capture*, K-CAP '19, Marina Del Rey, CA, USA: Association for Computing Machinery, 2019 243, ISBN: 9781450370080, URL: https://doi.org/10.1145/3360901.3364435 (cit. on pp. 94, 95).

[122] A. Oelen, M. Y. Jaradeh, K. E. Farfar, M. Stocker and S. Auer, "Comparing Research Contributions in a Scholarly Knowledge Graph", *Proceedings of the Third International Workshop on Capturing Scientific Knowledge co-located with the 10th International Conference on Knowledge Capture (K-CAP 2019), Marina del Rey, California , November 19th, 2019*, vol. 2526, CEUR Workshop Proceedings, CEUR-WS.org, 2019 21 (cit. on pp. 106, 135).

[123] V. Wiens and S. Lohmann, "Demonstration of a Customizable Knowledge Graph Visualization Framework.", *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020)*, 2020 (cit. on p. 110).

[124] H. Rijgersberg, M. van Assem and J. Top,
Ontology of units of measure and related concepts, Semantic Web **4**.1 (2013) 3 (cit. on p. 141).

[125] R. Cyganiak and D. Reynolds, *The RDF Data Cube Vocabulary*,
https://www.w3.org/TR/vocab-data-cube/, 2014 (cit. on p. 141).

Appendix

APPENDIX A

List of Publications

Conference Papers

1. **Vitalis Wiens**. *Volumetric Segmentation of Complex Bone Structures From Medical Imaging Data Using Reeb Graphs*, Central European Seminar on Computer Graphics for Students 2013, **3rd place in best paper awards**;
2. **Vitalis Wiens**, Lara Schlaffke, Tobias Schmidt-Wilcke, Thomas Schultz. *Visualizing Uncertainty in HARDI Tractography Using Superquadric Streamtubes* EuroVis (Short Papers) 2014;
3. Amin Abbasloo, **Vitalis Wiens**, Max Hermann, and Thomas Schultz. *Visualizing Tensor Normal Distributions at Multiple Levels of Detail*, IEEE Transactions on Visualization and Computer Graphics, 2015;
4. Shekoufeh Gorgi Zadeh, Maximilian WM Wintergerst, **Vitalis Wiens**, Sarah Thiele, Frank G Holz, Robert P Finger, and Thomas Schultz. *CNNs Enable Accurate and Fast Segmentation of Drusen in Optical Coherence Tomography*, Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support, 2017;
5. **Vitalis Wiens**, Steffen Lohmann, and Sören Auer. *Semantic Zooming for Ontology Graph Visualizations*, International Conference on Knowledge Capture, 2017;
6. **Vitalis Wiens**, Steffen Lohmann, Sören Auer. *GizMO–A Customizable Representation Model for Graph-Based Visualizations of Ontologies*. Proceedings of the 10th International Conference on Knowledge Capture 2019, 163–170, ACM.

7. Maximilian WM Wintergerst, Shekoufeh Gorgi Zadeh, **Vitalis Wiens**, Sarah Thiele, Steffen Schmitz-Valckenberg, Frank G Holz, Robert P Finger, and Thomas Schultz. *Replication and Refinement of an Algorithm for Automated Drusen Segmentation on Optical Coherence Tomography*, **Scientific Reports**, 2020;
8. **Vitalis Wiens**, Markus Stocker, Sören Auer. *Towards Customizable Chart Visualizations of Tabular Data Using Knowledge Graphs*. The 22nd International Conference on Asia-Pacific Digital Libraries (ICADL 2020).

Workshops and Demos

1. **Vitalis Wiens**, Steffen Lohmann, Sören Auer. *WebVOWL Editor: Device-Independent Visual Ontology Modeling*. International Semantic Web Conference 2018 (P&D); **Best Demo Award**
2. Amin Abbasloo, **Vitalis Wiens**, Tobias Schmidt-Wilcke, Pia C Sundgren, Reinhard Klein, and Thomas Schultz.
   Interactive Formation of Statistical Hypotheses in Diffusion Tensor Imaging, Eurographics Workshop on Visual Computing for Biology and Medicine, 2019;
3. **Vitalis Wiens**, Mikhail Galkin, Steffen Lohmann, Sören Auer. *Demonstration of a Customizable Representation Model for Graph-Based Visualizations of Ontologies–GizMO*. International Semantic Web Conference 2019 (Poster & Demos); **Best Demo Award**.
4. **Vitalis Wiens**, Steffen Lohmann. *Demonstration of a Customizable Knowledge Graph Visualization Framework*. ISWC 2020 (P&D).

APPENDIX B

Technical Aspects for a Pipeline-Based Approach in Semantic Web Contexts

We provide our pipeline-based approach as an open-source project with an MIT license[1]. In this appendix, we describe the technical aspects of the approach in more detail. Additionally, we provide examples for the configuration of example use cases.

Data Access Module

The data access module uses two customizable components, i.e., data access and parser components. When creating visualizations, we typically start with defining a data source for the data we wish to explore and comprehend. Since Semantic Web data is typically is distributed on the Web and can be accessed via URLs, SPARQL endpoints, or RESTful-APIs, the data access component specifies an endpoint URL that provides an entry point for obtaining the dataset.

Using the Object-Oriented-Programming conceptualization, this component is derived from a base component and overwrites its `__run__()` function which executes the corresponding code of the component. The result is consumed by other components as

[1] https://github.com/vitalis-wiens/donatello-pipelines

a parameter which value is obtained by calling the `getResult()` function of the data access component. Our conceptualizations expect the output to be provided in a JSON format. This requirement can be addressed in derived components of the data access component by providing additional request parameters to obtain the results in a JSON format, which most modern APIs support. In summary, the data access component executes code that obtains a JSON object and provides access to it via the `getResult()` function.

The foundation of the pipeline-based approach is the `Resource-Relation` model. After obtaining the JSON object from the data access component, we need to parse and map it to the `Resource-Relation` model. Since the JSON object's structure is a-priory not known, our parser component implements the transformation rules. These transformation rules have to be tailor-suited to a particular structure, i.e., the parser component overwrites the `__run()__` function to create the `Resource-Relation` model.

A general aspect of the `Resource-Relation` model is its textual reorganization of the input data into resources and relations with additional classifications of semantic elements into type assertions, annotations, and axioms. The `Resource-Relation` model provides two functions for integrating items into the model, i.e., `addResource(item)` and `addRelation(item)`. The `item` is either an instance of the class `Resource` or `Relation`. These classes provide additional functionality to store the URI, semantic types, and domain-range pairs for relations. Due to the aspect that the order of the triples can be arbitrary when obtaining results, the implementation of this model provides additional maps, whereas the keys are the resource/relation identifiers, i.e., URIs. These maps allow us to merge new incoming data to already defined resources or relations. Details about resources and relations are provided in Section 7.2. Our WYSIWIG pipeline configuration application provides examples of how to map results from VOWL-JSON, DBpedia SPARQL query results, Wikidata SPARQL query results, and RESTful API requests to the Open Research Knowledge Graph. In summary: The parser component creates a `Resource-Relation` model which contains resources and relations. The adjusted logic in the `__run()__` function has to call corresponding code to create resources/relations and integrate them into the model, e.g., `addResource(item)`.

Mapper Module

The `Resource-Relation` model is a simple reorganization of the textual representation of the obtained results. Similar as defined in RDF, the assignment into types, annotations, and axioms uses URIs or literal values without fully reflecting a graph structure. In order to create a visual representation, the `Resource-Relation` model requires further processing. Our approach uses two mapping steps in the mapper module to maintain its flexibility for different visual notations.

The first mapping generates a basic graph structure based on the `Resource-Relation` model. In particular, we create vertices and edges for resources and relations. Vertices and edges are created from the textual descriptions of resources and relations, respectively. However, the textual interlinking of elements using their URIs is now replaced by interlinking to corresponding vertices. In order to increase the flexibility and the selection of relevant information for the visualization, the `Vertex-Edge` mapper provides a definition map. This map defines value paths for resources and relations to look up particular values and assign them to vertices and edges. Furthermore, this mapper creates axiom edges for resources. While in its current implementation, this mapper does not provide additional axiom mappings for relational axioms, all items always have a reference to an element that is responsible for their creation. Thus, regardless of its current limitation, the information stored in resources and relations can be accessed in later steps in the visualization generation.

The second mapping transforms the `Vertex-Edge` model to a `Node-Link` model, which is used as input for the rendering module, responsible for the final visual representation. In order to keep the flexibility of the mapping and allow for the creation of various notations, the `Node-Link` mapper performs various graph manipulation operations (e.g., node splitting, node merging, and aggregations for nested representations such as UML). Similar to the `Vertex-Edge` mapper, this mapper uses a definition map to control the graph manipulations. The general aspect of node-link diagrams is that it consists of nodes and links. Nodes have a display name. Links have a display name and connect nodes. Thus, the main objective of the `Node-Link` mapper is to create nodes and links for the visualizations. Figure B.1 shows an example for the definition map that aggregates corresponding nodes and links addressing `owl:datatypeProperty` types for the realization of UML-based visualizations. The node mapper and link mapper objects define paths for accessing values from vertices and edges, respectively.

```
this.definitionMap = {
  nodeMapper: {
    __nodeType: "__vertexType",
    __nodeLinkIdentifier: "__vertexEdgeIdentifier",
    __displayName: "__displayName"
  },
  linkMapper: {
    __linkType: "__edgeType",
    __nodeLinkIdentifier: "__vertexEdgeIdentifier",
    __displayName: "__displayName",
    __linkAxiom: "__edgeAxiom",
    __source: "__source",
    __target: "__target"
  },
  nodeMerge: [{}],
  nodeSplit: [{}],
  aggregate: ["owl:datatypeProperty"],
  auxiliaryNodes: [] // here empty
};
```

Figure B.1: Definition map for a `Node-Link` mapper for the creation of a nested visualization.

Rendering Module

The rendering module provides four components. Their orchestration is responsible for the final interactive visual representation. In the following, we provide an overview of individual components, their conceptualizations, and their interplay.

Rendering Component

The rendering component implements the graph visualization and initializes other components. Since the graph consists of nodes and links, this component provides customizable node and link elements. Their visual appearance is defined using the descriptions provided by the rendering configuration objects. When nodes and links are created for the graph visualization, they receive a rendering configuration object. The particular configuration object is applied in the rendering function of an element by calling the `drawTools` component. The `drawTools` component receives an SVG group element to which the configured rendering elements are appended.

Since we use the separation of concern paradigm, other aspects of the final rendering are implemented in other components, which increases the flexibility and extensibility of the overall approach. The visual appearance of the graph is handled by the `renderingHandler` component. The spatial layout of elements is handled by the `layoutHandler` component. Interactions for graph, nodes, and links are implemented and configured in the `interactionHandler` component.

Visual Appearance

The visual appearance is defined by the `renderingHandler` component. The basic rendering handler provides a rendering configuration object which holds visual appearance configurations for nodes and links. Additionally this handler implements two functions, i.e., `getNodeConfigFromType` and `getNodeConfigFromType`. In our `Node-Link` model, nodes and links have a type that reflects their semantic type, e.g., `owl:Class` or `owl:ObjectProperty`. Thus, when creating a rendering primitive, its visual appearance configuration is fetched from the `renderingHandler` using the get function. The **drawTools** component creates rendering primitives based on these configurations. New visual appearance configurations can be created by creating new handlers that are derived from the basic rendering handler. New derived components have to overwrite the `renderingConfigObject` of the constructor.

The definition of the configuration objects for the visual appearance is inspired by GizMO [116]. However, we reorganize and group different aspects of the visual appearance definitions. A configuration object for nodes has three attributes, i.e., `style`, `fontStyle`, and `options`. The `style` configures the shape and the visual appearance of the rendering primitive. The `fontStyle` asserts the visual appearance of a label. The `options` object provides additional configurations, for example, to allow a label to dynamically adjust the shape size, i.e., `overwritesShapeSize`. An example for the configuration `owl:Class` OWL construct is provided in Listing B.1.

A configuration object for links has two attributes, i.e., `style`, and `options`. The `style` object has multiple attributes that control the visual appearance of links. The `link` object defines the general representation of a link, i.e., the line style (e.g., solid or dashed), line width, and line color. The `arrowHead` and `arrowTail` define the style of arrow-heads and tails, respectively. These have a `renderingType` (e.g., triangle or diamond) and additional rendering styles for colors and strokes. The `propertyNode` shares the definitions with nodes, having `style`, `fontStyle`, and `options` attributes.

The link attribute `options` object defines the rendering type of a link, i.e., a straight line or a curve. Additionally, it specifies if arrow-head, arrow-tail, and property node should be rendered. An example for the configuration `owl:DatatypeProperty` OWL construct is provided in Listing B.2.

```
"owl:Class": {
      style: {
        renderingType: "circle",
        bgColor: "#aaccff",
        strokeElement: true,
        strokeWidth: "1px",
        strokeStyle: "solid",
        strokeColor: "#000",
        radius: 50,
        width: 100,
        height: 50
      },
      fontStyle: {
        fontFamily: "Helvetica,Arial,sans-serif",
        fontColor: "#000",
        fontSize: "12px"
      },
      options: {
        drawDisplayName: true,
        drawNestedAttributes: false,
        cropLongText: false,
        addTitleForDisplayName: true,
        overwritesShapeSize: false,
        overwriteOffset: 0,
        fontPositionH: "center",
        fontPositionV: "center"
      }
    }
```

Listing B.1: Example for node configuration defining visual appearance of owl:Class.

```
"owl:datatypeProperty": {
  style: {
    link: { lineStyle: "solid", lineWidth: "2px",
        lineColor: "#000000"
    },
    arrowHead: { renderingType: "triangle",
          scaleFactor: 1, strokeWidth: "2px",
          strokeStyle: "solid", strokeColor: "#000000",
          fillColor: "#000000"
    },
    propertyNode: {
      style: { renderingType: "rect",
        bgColor: "#99CC66", roundedCorner: "0,0",
        fontSizeOverWritesShapeSize: "true",
        overWriteOffset: "5", strokeElement: "false",
        radius: 50, width: 100, height: 25
      },
      fontStyle: { fontFamily: "Helvetica,Arial,sans-serif",
        fontColor: "#000000", fontSize: "12px"
      },
      options: {drawDisplayName: true,
        cropLongText: true, addTitleForDisplayName: true,
        overwritesShapeSize: false,overwriteOffset: 0,
        fontPositionH: "center", fontPositionV: "center"
      }
    }
  },
  options: {
    drawPropertyNode: true,
    drawArrowHead: true, drawArrowTail: false,
    link_renderingType: "line" // line or curve
  }
}
```

Listing B.2: Example for link configuration defining visual appearance of owl:DatatypeProperty.

Spatial Layout

The spatial layout component asserts the positions or rendering elements in the graph. Our current implementation provides a single layout algorithm using D3.js force layout. The constructor of this component receives the graph as a parameter to obtain the rendering elements, i.e., nodes and links. The implementation of this component is tailor-suited to the requirements of the library and its parameters. Furthermore, our implementations provide functions that assert configuration parameters and allow the rendering component to control the behavior of the force layout. Additionally, the interactions component has a reference to the layout handler to provide control mechanisms based on interactions with graph, nodes, or links. For example, the drag interaction of a node calls the function `resumeForce()` of our force layout component to enable dynamic rearrangement of other rendering elements in the graph.

The implementation of other layout algorithms has to comply with the chosen library specifications. Furthermore, when creating new layout handlers, we have to ensure that these provide functions that can be called from other components. In particular, we have to enable user interactions within the graph or GUI features that invoke functions of the newly created layout handler.

Interactions

User interactions with the graph, the nodes, and the links are provided in the interactions handler. Following the separation of concern paradigm, the interaction handler has three components for interactions with the graph, nodes, and links. The interactions handler is instantiated in the rendering component. The individual interaction components are initialized in this component. Furthermore, we provide additional checks if these have been initialized by a derived component. If no specific interaction components are defined in a derived component, the default fallback interactions are instantiated and applied to the graph, nodes, and links, respectively.

Graph Interactions: The graph interactions specify interactions with the canvas area. The basic interactions include drag and pan interactions using the mouse. Additionally, mouse wheel zoom events are applied. These zoom events also implement smooth animation transformation for these types of interactions.

Node Interactions: Node interactions are injected into the rendering primitives that are provided by the rendering component. The basic interactions include drag, hover, click and double-click interactions. The node interactions are injected on the root rendering primitive of an element. Our current conceptualization requires the rendering primitives to be rendered first. The injected events will then exist on the rendered DOM elements. However, our conceptualization allows for flexible extensions of interactions decoupled from the implementation of nodes rendering logic. While the events are registered to particular instances of nodes, the executing functions are implemented in the interactions handler for the nodes. These functions have access to the rendering component and the data of the particular node. Thus, shifting the responsibility of interactions implementations and the resulting behavior to this component. When nodes are instantiated, they also receive references to the graph and the layout handler. Using these references, node interactions can invoke function calls on the graph or the layout handler, e.g., requesting node collapse or expand operations in the graph or resume a force layout algorithm to create dynamic layouts when a node is dragged in the canvas area.

The ORKG node interactions implementation showcases an example where we use node hover interactions to render additional buttons into a node. In our conceptualizations rendering primitives have an SVG root group to which new elements can be appended. However, we argue that the logic of the new interactions has to remain in the particular interactions handler, making it reusable for other use-cases. Nevertheless, the provided implementation of the rendering component and the layout handler have to comply with the newly created interaction. Thus, we suggest creating rendering modules that cover the implementation of all components for individual use cases. This will ensure that rendering modules can be switched out effortlessly and provide additional examples and reusable components for other use cases.

Link Interactions: Similar to node interactions, this component provides the implementation for link interactions. However, we make here a distinction between the rendered line that connects two nodes and the `propertyNode`. The `propertyNode` is a rendering primitive that renders the label of a link. It is derived from nodes. Thus, we can apply a similar implementation of its interactions as in the node interactions component. The injection of events into the DOM elements is two-fold. For example, we can inject drag and click events for the `propertyNode`.

Summary

The rendering module handles different aspects of interactive visual representations using the separation of concern paradigm. Our WYSIWYG pipeline builder prototype provides various examples for different use cases and interactions. Our general suggestion to create complete rendering modules for individual use cases ensures that the final visualization is implemented as intended by developers. While a visual representation may be completely tailor-suited to a particular use case, through the separation of concern paradigm, individual components for visual appearance, spatial layouts, and interactions can be modified and adapted to other use cases. Furthermore, these can be used as building blocks and code examples for the creation of other visual representations.

Curriculum Vitae

Personal Details

Name	Vitalis Wiens
Date of Birth	02.09.1986
Email	vitalis.wiens@gmail.com

Education

1998–2007	Abitur, Konrad-Adenauer-Gymnasium Meckenheim, Germany
2007–2010	Bachelor Studium in Elektrotechnik, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, Germany.
2010–2011	Bachelor Studium in Informatik, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, Germany.
2011–2015	BSc in Informatik Rheinische Friedrich-Wilhelms-Universität, Bonn, Germany.
2015–2017	MSc in Informatik Rheinische Friedrich-Wilhelms-Universität, Bonn, Germany.
2017–2019	Doktorand, Stipendium TIB & Wissenschaftlicher Mitarbeiter bei FraunhoferIAIS (Enterprise Infromation Systems, St. Augustin, Germany).
2019–2021	Doktorand, Wissenschaftlicher Mitarbeiter bei LUH (L3S,TIB), Hannover, Germany.
2021–jetzt	Research Engineer bei metaphacts, Walldorf, Germany.

Vitalis Wiens

vitalis.wiens@gmail.com
github.com/vitalis-wiens

Experience

Research Assistant

L3S & TIB Joint Lab, Front End Developer (JavaScript/React) 2019–2021
Knowledge Graph visualization – Supervisor: Prof. Dr. Sören Auer & Dr. Markus Stocker

- Improved user experience by layout customizations and graph animations
- Improved network load by caching and reusing information

Fraunhofer IAIS (Enterprise Information Systems), Full Stack Developer (Java/JavaScript/Docker) 2017–2019
Sensor data visualization as graphs and visual data modeling – Supervisor: Dr. Steffen Lohmann

- Successfully deployed industry project with a car manufacturer by restructuring information flow using CAN bus synchronization methods and adjusting backend processing for visualization generation
- Improved user experience in an EU project by visual data modeling extension

Student Assistant

Fraunhofer IAIS (Enterprise Information Systems), Front End Developer (JavaScript) 2016–2017
Ontology visualization and interaction – Supervisor: Dr. Steffen Lohmann

- Improved user interface by introducing collapsible sidebar, processing progress bar, and a scrollable navigation menu (optimized for mobile devices)
- Improved graph searching by dictionary lookups, halo visualization, and location function

Institute of Computer Graphics (University of Bonn), Developer (C/C++/Qt/OpenGL/Python) 2011–2016
Diffusion tensor imaging to visualize brain structures – Supervisor: Prof. Dr. Thomas Schultz

- Extended DTI framework for fiber visualizations with superquadric streamtubes
- Improved dataset processing time by implementing multithreaded processing

Industry project for hair simulation – Supervisor: Prof. Dr. Andreas Weber

- Designed approach for creating intersection free geometry structures of fibers

Bone segmentation tasks of µCT scans of rodent skulls – Supervisor: Dr. Max Hermann

- Designed batch processing tool for parameter tuning in bone segmentation tasks using Qt and SQLite, which improved evaluation experiment reproducibility

Institute of Human-Machine Interaction (RWTH Aachen), Developer (C++/Qt) 2010–2011
Export of 3D models from Autodesk 3ds Max – Supervisor: Prof. Dr. Christian Schlette

- Increased runtime performance by 80% by optimizing code and introducing multithreaded processing

Institute of Operating Systems (RWTH Aachen), Tutor 2009–2010
Support teaching of computer science courses – Supervisor: Dr. Stefan Lankes

- Transformed x86 assembler script to NASM and designed exercise for students
- Supervised students in a C++ object-oriented programming lab course

Education

PhD Candidate Computer Science, Leibniz University Hannover, Germany 2017–Present

- **Advisor:** Prof. Dr. Sören Auer
- **Thesis topic**: “Visual Exploration of Semantic-Web-Based Knowledge Structures”
- **Fields of interest**: Scientific Visualization, Semantic Web
- **Research Projects:**
  - **Open Research Knowledge Graph**
    Representing the content of scientific articles as a Knowledge Graph
    Contribution: Facilitate understanding of the Knowledge Graph by developing novel exploration methods and customizable visual representations
  - **Donatello**
    Tool for creating customizable Knowledge Graph visualization pipelines
    Contribution: Provide an easily configurable and extendable codebase for creating interactive visualizations of different Semantic Web data sources
  - **GizMO**
    Customizable visualizations for ontologies using node-link diagrams
    Contribution: Describe the visual appearance and spatial layouts for node-link diagrams as OWL ontologies, allowing to attach visual appearance and spatial positioning to ontologies without modifying them

M.Sc. Computer Science, University of Bonn, Germany 2015–2017

- **Advisor:** Prof. Dr. Sören Auer & Dr. Steffen Lohmann
- **Focus**: Computer Graphics, Scientific Visualization
- **Thesis**: “Ontology Graph Visualizations Enhanced with Semantic Zooming”
- **GPA**: 3.75 / 4

B.Sc. Computer Science, University of Bonn / RWTH Aachen University, Germany 2010–2015

- **Advisor:** Prof. Dr. Thomas Schultz
- **Focus**: Computer Graphics, Scientific Visualization
- **Thesis**: “Superquadric Streamtubes for Fiber Visualization in Diffusion Imaging”
- **GPA**: 3.1 / 4

B.Sc. Electrical Engineering, RWTH Aachen University, Germany 2007–2010

- **Focus**: Technical Computer Science

Skills

- **Proficient with**
  ⬡ JavaScript (ES6) ⬡ React / Redux ⬡ C++ ⬡ CSS3 ⬡ HTML5 ⬡ D3.js
  ⬡ Qt4 ⬡ Git ⬡ Linux ⬡ LaTeX ⬡ Semantic Web (RDF, OWL, Knowledge Graphs)
- **Skilled with**
  ⬡ Java ⬡ Python 3 ⬡ SPARQL ⬡ Docker ⬡ VTK ⬡ Blender
- **Familiar with**
  ⬡ OpenGL ⬡ SQL ⬡ Assembler (x86) ⬡ Matlab ⬡ OpenCV
- **Natural Languages**
  ⬡ German (Native) ⬡ English (Proficient) ⬡ Russian (Intermediate)

Teaching Experience

▢ **Supervised** 11710 “Knowledge Engineering and Semantic Web” at LUH as tutor	2020
▢ **Supervised** MA-INF 4312 “Semantic Data Web Technologies” at University of Bonn as tutor	2019, 2018
▢ **Co-supervised** master thesis for a device-independent ontology editor at University of Bonn	2017
▢ **Supported** BA-INF 122 “Scientific Visualization” at University of Bonn with coding exercises	2016, 2015
▢ **Supervised** 61.61.06119 “Fundamentals of Computer Engineering 4 ” at RWTH as tutor	2009

Academic Activities

▢ **Co-organizing** Visualization and Interaction for Ontologies and Linked Data Workshop (Voila)	2021
▢ **PC member** at International Conference on Business Information Systems (BIS21)	2021
▢ **Co-organized** Visualization and Interaction for Ontologies and Linked Data Workshop (Voila)	2020
▢ **PC member** at International Conference on Knowledge Capture (K-CAP)	2019, 2017
▢ **Co-organized** Computer Science Conference for University of Bonn Students (CSCUBS)	2019
▢ **Reviewer** for Workshop on Ontology Design and Patterns (WOP)	2019
▢ **PC member** at The Joint Ontology Workshops (JOWO)	2019
▢ **Reviewer** for Extended Semantic Web Conference (ESWC)	2019, 2018
▢ **Reviewer** for Semantics	2018
▢ **Reviewer** for International Conference on Semantic Computing (ICSC)	2018
▢ **Reviewer** for International Semantic Web Conference (ISWC)	2017

Awards

- ▢ **Best Demo Award** at International Semantic Web Conference, Poster and Demos, 2019 – **GizMO**🔗 ▶
- ▢ **Best Demo Award** at International Semantic Web Conference, P&D, 2018 – **WebVOWL Editor**🔗 ▶
- ▢ **3rd Best Paper Award** at Central European Seminar on Computer Graphics for Students 2013🔗