Patrick Lambrix
Catia Pesquita
Vitalis Wiens (Eds.)

VOILA! 2021

Proceedings of the 6th International Workshop on

Visualization and Interaction for Ontologies and Linked Data

Co-located with ISWC 2021, Virtual, October 25, 2021.

Title: Visualization and Interaction for Ontologies and Linked Data (VOILA! 2021)

Editors: Patrick Lambrix, Catia Pesquita, Vitalis Wiens

ISSN: 1613-0073

CEUR Workshop Proceedings
(CEUR-WS.org)

Organizing Committee

Patrick Lambrix, Linköping University and University of Gävle, Sweden
Catia Pesquita, LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
Vitalis Wiens, L3S, TIB, & Leibniz University Hanover, Germany

Program Committee

Kārlis Čerāns, University of Latvia, Latvia
Aba-Sah Dadzie, Edinburgh University, UK
Anastasia Dimou, KU Leuven, Belgium
Roberto García, Universitat de Lleida, Spain
Alain Giboin, Université Côte d'Azur, Inria, CNRS, I3S, France
Anika Groß, University of Leipzig, Germany
Ali Hasnain, Royal College of Surgeons Ireland, University of Medicine and Health Sciences
Emmanuel Pietriga, INRIA Saclay, France
Harald Sack, FIZ Karlsruhe, Leibniz Institute for Information Infrastructure & KIT Karlsruhe, Germany
Daniel Schwabe, Pontifical Catholic University of Rio de Janeiro, Brazil
Kamran Sedig, University of Western Ontario, Canada
Ahmet Soylu, Norwegian University of Science and Technology, Norway
Markel Vigo, University of Manchester, UK

Preface

The Semantic Web enables intelligent agents to create knowledge by interpreting, integrating and drawing inferences from the abundance of data at their disposal. It encompasses approaches and techniques for expressing and processing data in machine-readable formats. All these tasks demand a human-in-the-loop; without them, the great vision of the Semantic Web would hardly be achieved. Meanwhile, visual interfaces for modeling, editing, exploring, integrating, etc., of semantic content have not received much attention yet.

The size and complexity of ontologies and Linked Data in the Semantic Web constantly grows and the diverse backgrounds of the users and application areas multiply at the same time. Providing users with visual representations and intuitive interaction techniques can significantly aid the exploration and understanding of the domains and knowledge represented by ontologies and Linked Data.

Ontology visualization is not a new topic and a number of approaches have become available in recent years, with some being already well-established, particularly in the field of ontology modeling. In other areas of ontology engineering, such as ontology alignment and debugging, although several tools have recently been developed, few provide a graphical user interface, not to mention navigational aids or comprehensive visualization and interaction techniques.

In the presence of a huge network of interconnected resources, one of the challenges faced by the Linked Data community is the visualization of multidimensional datasets to provide for efficient overview, exploration and querying tasks, to mention just a few. With the focus shifting from a Web of Documents to a Web of Data, changes in the interaction paradigms are in demand as well. Novel approaches also need to take into consideration the technological challenges and opportunities given by new interaction contexts, ranging from mobile, touch, and gesture interaction to visualizations on large displays, and encompassing highly responsive web applications.

There is no one-size-fits-all solution but different use cases demand different visualization and interaction techniques. The evaluation of such interfaces and techniques poses another relevant concern given the specific challenges of visualizing data imbued with semantic complexity. Ultimately, providing better user interfaces, visual representations and interaction techniques will foster user engagement and likely lead to higher quality results in different applications employing ontologies and proliferate the consumption of Linked Data.

These and related issues are addressed by the VOILA! workshop series concerned with *Visualization and Interaction for Ontologies and Linked Data*. The sixth edition of VOILA! was co-located with the 20th International Semantic Web Conference (ISWC 2021) and took place as a half-day virtual event on October 25, 2021. It was organized around scientific paper presentations and discussions.

The call for papers for VOILA! 2021 attracted 13 submissions in different paper categories. At least three reviewers were assigned to each submission. Based on the reviews, we selected 10 contributions for presentation at the workshop.

We thank all authors for their submissions and all members of the VOILA! program committee for their useful reviews and comments. We are also grateful to Laura Hollink and Mayank Kejriwal, the workshop chairs of ISWC 2021, for their continuous support during the workshop organization.

October 2021

Patrick Lambrix,
Catia Pesquita,
Vitalis Wiens

Contents

A First Step towards a Tool for Extending Ontologies

Mina Abd Nikooie Pour[1], Huanyu Li[1,3],
Rickard Armiento[2,3], and Patrick Lambrix[1,3,4]

[1] Department of Computer and Information Science, Linköping University, Sweden
[2] Department of Physics, Chemistry and Biology, Linköping University, Sweden
[3] The Swedish e-Science Research Centre, Linköping University, Sweden
[4] Department of Building Engineering, Energy Systems and Sustainability Science, University of Gävle, Sweden
firstname.lastname@liu.se

Abstract. Ontologies have been proposed as a means towards making data FAIR (Findable, Accessible, Interoperable, Reusable). This has attracted much interest in several communities and ontologies are being developed. However, to obtain good results when using ontologies in semantically-enabled applications, the ontologies need to be of high quality. One of the quality aspects is that the ontologies should be as complete as possible. In this paper we propose a first version of a tool that supports users in extending ontologies using a phrase-based approach. To demonstrate the usefulness of our proposed tool, we exemplify the use by extending the Materials Design Ontology. [1]

1 Introduction

In many areas there is a recent interest in making data FAIR, i.e., Findable, Accessible, Interoperable, and Reusable [10]. Findable refers to the fact that data and metadata should be easy to find, accessible to the fact that it should be clear how to access the data, interoperable to the fact that the data needs to be integrated with other data and be usable by applications and workflows, and reusable to the fact that data and metadata are well described such that the data can be replicated or combined in different settings. Ontologies can alleviate some of the issues towards making data FAIR, such as in ontology-based access to multiple data sources for Interoperability and using ontologies to describe data source capabilities for Findability.

Ontologies need to be of high quality. One of the quality aspects is that the ontologies should be as complete as possible.[2] If an ontology is not complete,

[2] This relates to the requirement of domain coverage in [9]. In practice, knowing when an ontology is complete is difficult but it is possible to define an 'is more complete than' relation between ontologies which can be used for comparing completeness [5].

it may lead to missing inferences and thus incomplete results in semantically-enabled applications.

Many techniques exist for finding missing information in ontologies and extending them. The ontology extension problem that we tackle deals mainly with concept discovery and concept hierarchy derivation which are also two of the tasks in the problem of ontology learning [3]. Therefore, there are many techniques that can be used as shown by a recent survey [2] that discusses 140 research papers. These techniques are usually linguistics-, statistics-, or logic-based. One such technique is phrase-based extension where frequent phrases are collected from text and proposed to a domain expert as candidates for creating concepts with similar names. An example of such a method that has been applied to ontologies in the materials science field is found in [6,1]. When using the method the phrases were presented in an Excel file to the domain expert who then annotated the file with the concepts and axioms to add to the ontology. Then this information was transferred into OWL format manually or via an ontology editor. It was clear that user support for these tasks was needed. Therefore, in this paper we present a first version of a tool that takes as input a list of phrases (generated using the method in [6,1] or any other method that outputs phrases) and supports the user in turning phrases into concepts and in defining axioms related to the new concepts. We show the use of the tool in a use case where we extend the Materials Design Ontology.

The remainder of the paper is organized as follows. In the background section we describe the method for ontology extension on which our proposed tool is based (sub-section 2.1), as well as the ontology that we extend to exemplify the method (sub-section 2.2). In section 3 we briefly discuss the purpose, intended users and development of the tool. We describe our tool in section 4. In section 5 we show a use case in the materials science domain where we extend the Materials Design Ontology and briefly discuss a user interface evaluation. The paper concludes in section 6.

2 Background

2.1 Method for extending ontologies

Our tool is based on the approach for extending ontologies presented in [6] and extended in [1]. Figure 1 gives an overview.

Given a corpus of unstructured text, in a first step a phrase-based topic model is produced. The output of this step is a list of frequent phrases as well as a number of topics with representative phrases. In the method described in [6,1] this is done using the ToPMine system [4]. The approach in [6] uses a formal topical concept analysis-based approach to mine topics. The frequent phrases, the topics, and the result of the formal topical concept analysis are suggestions that a domain expert should validate or interpret and relate to concepts in the ontology.

The method in [6] introduced different categories for the phrases related to processing that could lead to the ontology extension based on the phrases. The

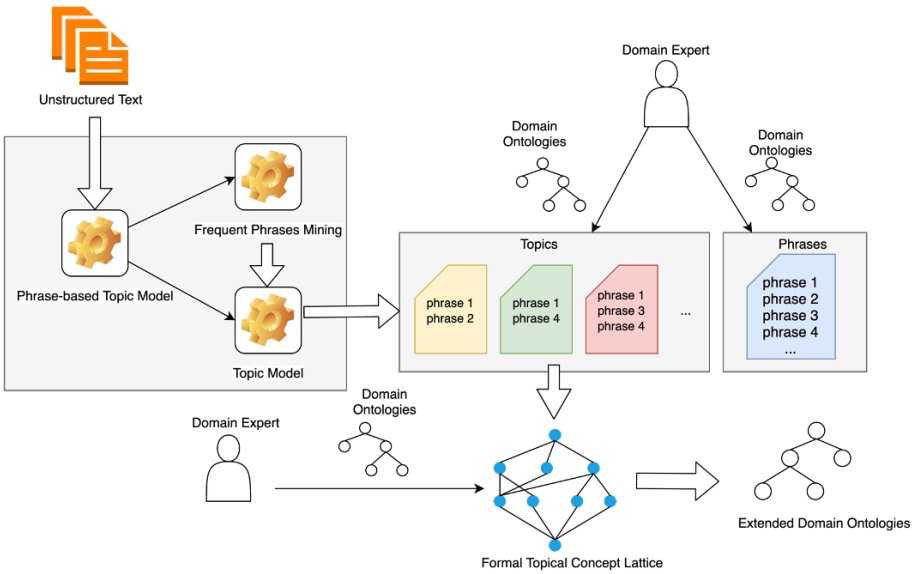


Fig. 1. Approach: The upper part of the figure shows the creation of a phrase-based topic model with unstructured text as input and phrases and topics as output. The lower part shows the formal topical concept analysis with as input topics and as output a topical concept lattice. In both parts a domain expert validates and interprets the results. [6].

ADD category represents the addition of concepts where the new concept has the same name as a phrase. For the ADD-m category a new concept is added with a name that is a modification of a phrase. The EXIST and EXIST-m categories do not represent a change, but indicate that there already is a concept in the ontology with the same name as the phrase or a modified version of the phrase, respectively. The No category represents that the phrase cannot be used to create a concept for the ontology, while the No-g category represents the fact that the phrase should not lead to a new concept in this ontology, but could be used in a more general ontology. We also introduce here NEW which represents the addition of a new concept that is not related to a phrase. We note that this categorization is interesting for experimenting and gathering data regarding the kinds of operations a user performs when extending an ontology, but not necessarily for an end user using the tool.

During the work on extending ontologies using this approach, it became clear that the domain and knowledge engineering experts needed tool support. In this paper we have implemented a tool for helping a domain expert validating phrases and creating concepts and axioms related to these phrases in the ontology. The validation of topics and axioms related to topics is left for future work. Large parts of the current functionality related to phrases will be reusable for topics, but some new visualizations will be needed.

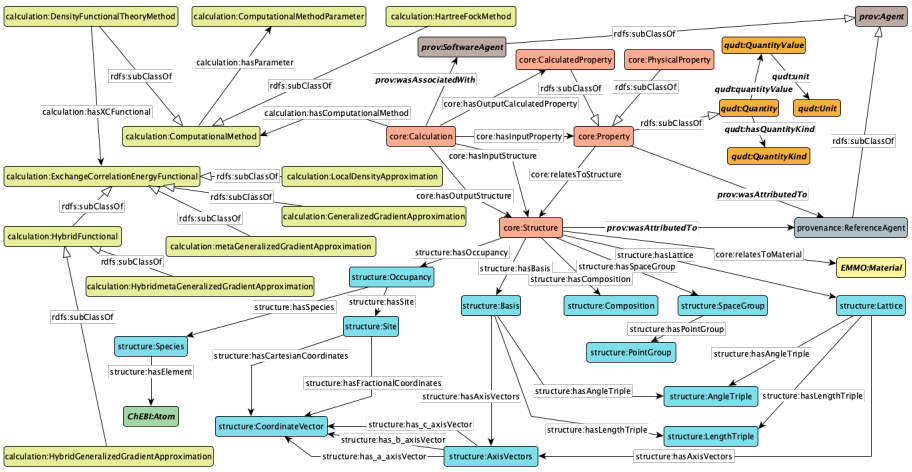


Fig. 2. The Materials Design Ontology [7]. The different modules *Core*, *Structure Calculation* and *Provenance* as well as other connected ontologies have different colors.

2.2 Materials Design Ontology

The ontology that we use to exemplify our approach is the Materials Design Ontology (MDO) presented in [7] and available at `https://w3id.org/mdo`. MDO is an ontology covering basic concepts for materials design (Figure 2). The development was guided by the schemas of the Open Databases Integration for Materials Design (OPTIMADE[3]) project. The OPTIMADE project aims at making materials databases interoperable by developing a common API and the consortium includes many of the important database providers in the field. Therefore, the OPTIMADE schemas are based on a consensus reached by several of the important stakeholders in the field. The aim is to use MDO for providing semantic search and integration of materials databases, e.g., via OPTIMADE.

3 Requirements and Development

Before we present our tool in section 4, we discuss the system requirements to clarify some of its aspects and use the dimensions of the design space for user studies in Semantic Web contexts in [8] as guidelines.

Purpose. The purpose of the tool is to support a user in extending an ontology using a phrase-based approach. The tool supports the user in creating concepts from phrases as well as in adding subsumption axioms related to the new concepts. The main purpose of the tool is *creating and managing* as defined in [8] as it is essentially an ontology authoring tool that supports the creation of new concepts and axioms for an ontology. However, there is also an aspect

[3] `https://www.optimade.org/`

of *learning and understanding* as a user needs to understand the phrases, the concepts and their contexts to be able to make informed decisions about which phrases can lead to concepts and axioms that should be added to the ontology. Therefore, in addition to supporting tasks related to creating, adding and managing concepts and axioms based on phrases, also tasks related to exploration and searching for information in the frequent phrases and ontology need to be supported.

Users. The tool is supposed to be used by users or teams of users that have skills and experience in two fields. As it is an ontology authoring tool the user team needs to be well-versed in ontology development and understand such things as what an ontology can contain, how reasoning works and what the consequences are of adding an axiom to an ontology. Further, the user team also needs expertise in the domain of the ontology, e.g., in our use case exemplifying the use of the tool, the user team needs expertise in materials science.

Development. The tool was developed in an agile way, with meetings about once a week with ontology engineering and materials science experts. This allowed for fast feedback and the discussions led to changes in the design choices. Some examples of changes were the use of tabs instead of windows, the addition of the possibility to mark phrases as finalized, and the possibility to add additional concepts when defining a new concept. One item that we have currently kept in the tool is the ability to classify the type of action (using ADD, ADD-m, EXIST, EXIST-m, No, No-g, see explanation in section 2.1). In the final version of the tool for end users this is not needed and the parts of the user interface referring to this will be removed, but while experimenting with the tool this gives us valuable information.

4 Ontology Extension System

In this section we describe our tool with its functionality and current user interface. We use screenshots from extending MDO. We decided to model the interface in a way that reflects the workflow of the user. There are three main steps: *Set-up* where ontology and phrases are loaded, *From Phrase to Concept* where phrases are used to define concepts, and *From Concept to Axiom* where axioms related to the concepts are defined. Each of the steps in the workflow is represented by a tab in the tool. The workflow is iterative and the steps are repeated for different phrases and concepts.

4.1 Set-up

The first step in the workflow is loading the necessary files including the ontology that we want to extend and the frequent phrases. In Figure 3 MDO has been loaded from OWL files (as four different modules) and the frequent phrases have been loaded from a file that was generated earlier.

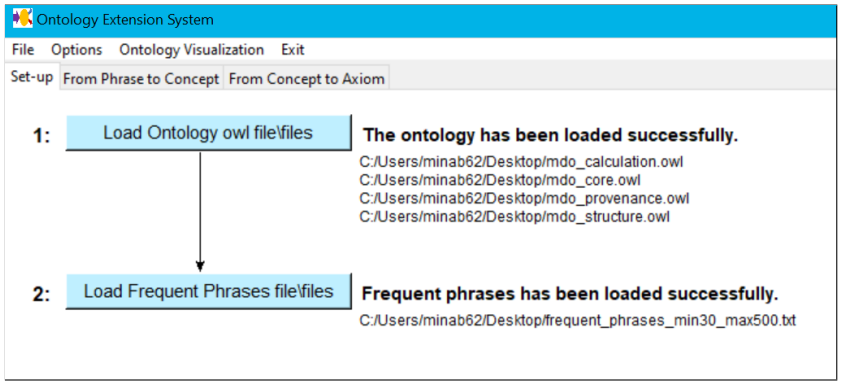


Fig. 3. The *Set-up* tab of the Ontology Extension System.

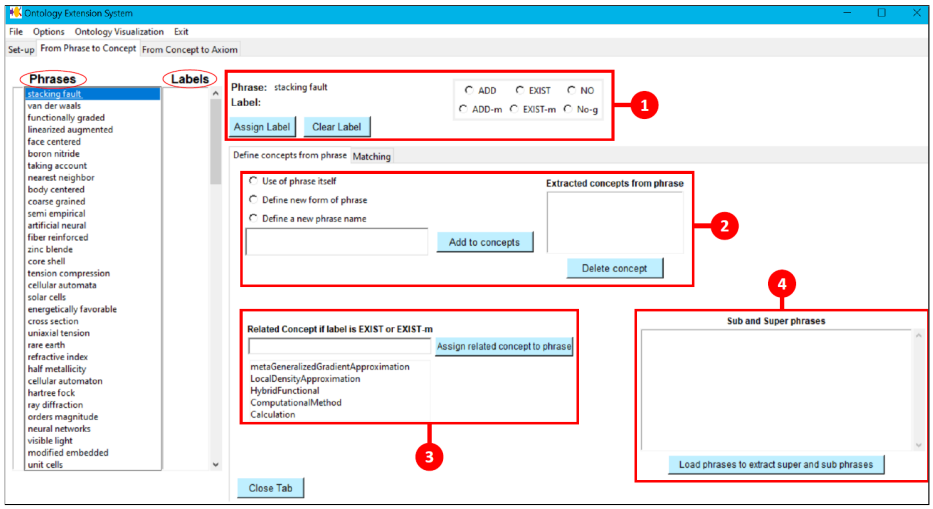


Fig. 4. The *From Phrase to Concept* tab of the Ontology Extension System.

4.2 From Phrase to Concept

In this step of the workflow the user processes the frequent phrases and decides for each phrase whether one or more concepts related to the phrase can be defined in the ontology. Figure 4 shows the tab for this phase.

At the left hand side of the *From Phrase to Concept* tab two main columns, *Phrases* and *Labels* are shown. The *Phrases* column contains the frequent phrases loaded or generated in the previous step. The *Labels* column will contain one or more of ADD, ADD-m, EXIST, EXIST-m, No, No-g in case the phrase has been (partly) processed through assigning label functionality.

By clicking on a phrase in the *Phrases* column, inner tabs are opened that support the processing of the phrase.

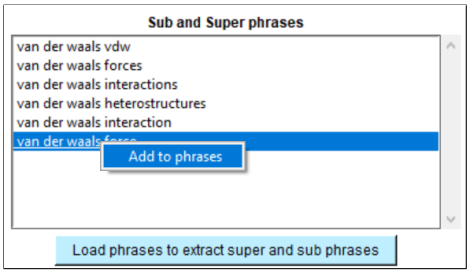


Fig. 5. Extracting sub- and super-phrases from another frequent phrases list.

In the part marked by red frame 2 in Figure 4 the user can define a new concept for the ontology. There are three options represented by the three radio buttons. For the name of the new concept the phrase can be used as is (ADD, first button), or a modified version of the phrase can be used, e.g., removing plural 's' (ADD-m, second button). The third button allows to define a new unrelated concept which was deemed necessary by the domain experts (NEW). Further, all concepts created in relation to the phrase are shown in the text box on the right side of the frame.

The part marked by red frame 1 in Figure 4 is mainly interesting for our experiments and allows a user to directly assign or remove categorization labels.

In the part marked by red frame 3 in Figure 4 the user can relate an existing concept to the phrase and implicitly assign an EXIST or EXIST-m label.

The part related to red frame 4 in Figure 4 allows for searching sub- and super-phrases in other lists. This is useful, for instance, when different frequent phrase lists are generated by a system with different parameter settings. This functionality allows to also use parts of the other lists by adding these phrases as well. An example is shown in Figure 5.

In the *Phrases* list, the phrases can be marked or unmarked as processed by right-clicking on the desired phrase. This allows the user to mark progress. An additional help, using the *Matching* functionality, is to search for phrases or concepts with a name similar to a phrase. The algorithm uses fuzzy matching on the phrases and concepts and a string matcher on their stemmed version. This information could be used, for instance, to relate concepts and phrases or as extra information to the decision process regarding adding new concepts based on phrases.

4.3 From Concept to Axiom

Figure 7 shows the tab related to the next step in the workflow for extending ontologies using the phrase-based approach. In this step axioms related to the new concepts are added to the ontology.

At the left hand side of the tab there is a column of concepts which consists of concept names from the ontology loaded during the *Set-up* phase together with the ones that have been added by the user in the *From Phrase to Concept*

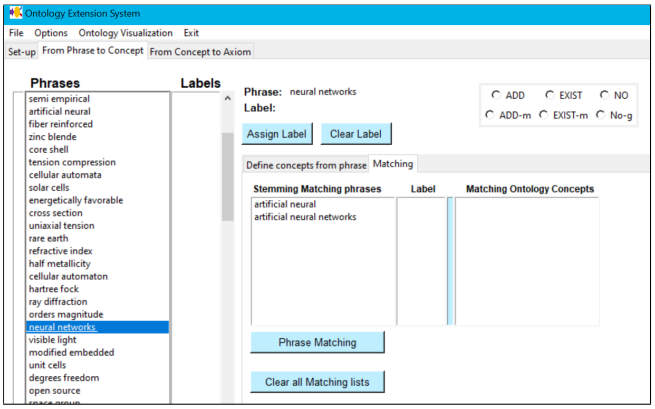


Fig. 6. Matching a phrase with other phrases and concept names.

phase. The origin relates a new concept to the phrase that was used for defining the concept.

In the upper part of the tab, axioms can be defined. By clicking on a concept in the concept list, the concept name appears at two places with radio buttons to help the user define sub- and super-concepts of the concept. These can be selected from the list shown in the text box shown on the right side.

The existing axioms as well as the axioms defined by the user are displayed in the bottom part of the tab in the list labeled as *Axioms defined for the concept*. To provide more information regarding the context of an axiom, clicking on an axiom shows the axioms related sub-concepts of the concept in the left-hand side of the axiom and super-concepts of the concept in the right-hand side of the axiom. An example is given in Figure 8.

4.4 Additional functionality

In addition to the tabs described earlier other functionalities have been implemented as shown in the menu bar. The **File** menu contains functionalities related to creating, opening and saving files. For instance, the lists of phrases and concepts can be saved as Excel files including added parts such as defined axioms, which phrase was the origin for a concept, assigned labels and concepts created for a phrase. The extended ontology can be saved as an OWL file. We note that partial results can be saved and loaded again at a later time. The **Options** menu contains functionality to search phrases, concepts and axioms. Additionally, all the axioms (Figure 9) and equivalent concepts can be displayed as well. **Ontology Visualization** is used to visualize the ontology as well as the changes made. The new concepts and axioms are displayed in a different color from the concepts and axioms that were in the original ontology (Figure 10).

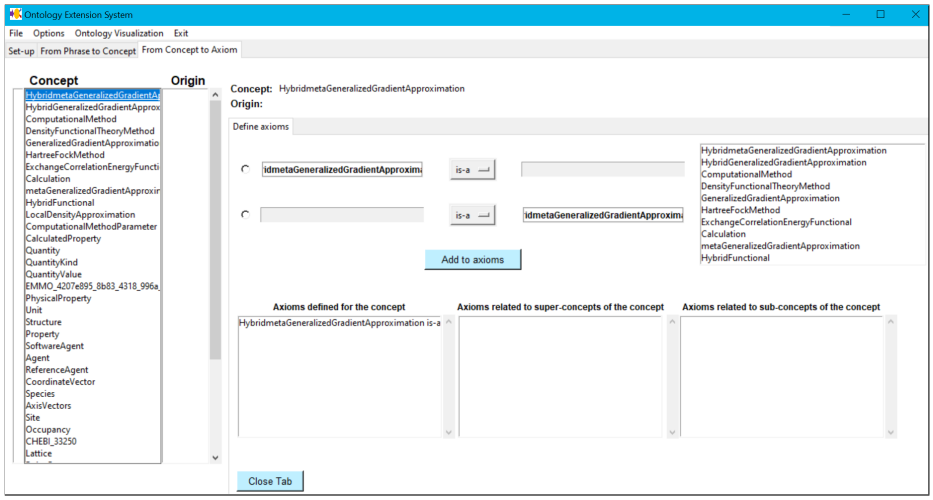


Fig. 7. The *From Concept to Axiom* tab of the Ontology Extension .

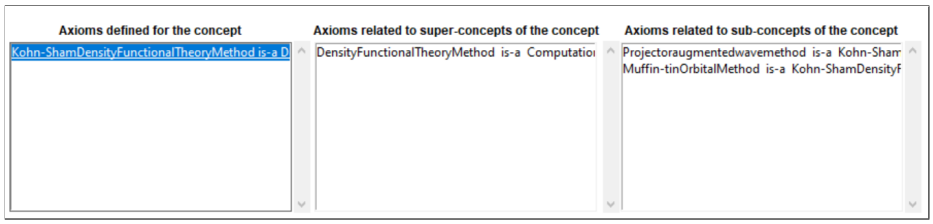


Fig. 8. Axioms related to sub- and super-concepts of concepts in an axiom.

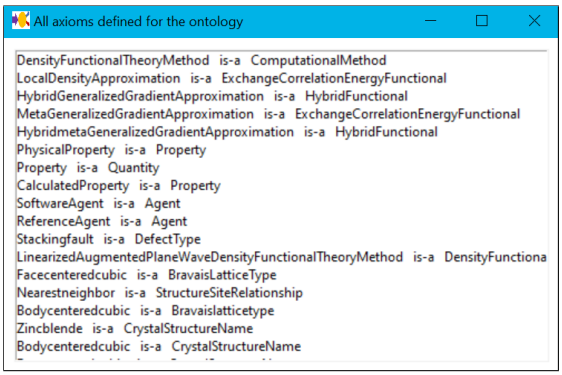


Fig. 9. Axioms of the ontology.

5 Experiments

We performed a feasibility study to evaluate the functionality and usefulness of our tool by extending MDO. Furthermore, we performed a small evaluation of the user interface.

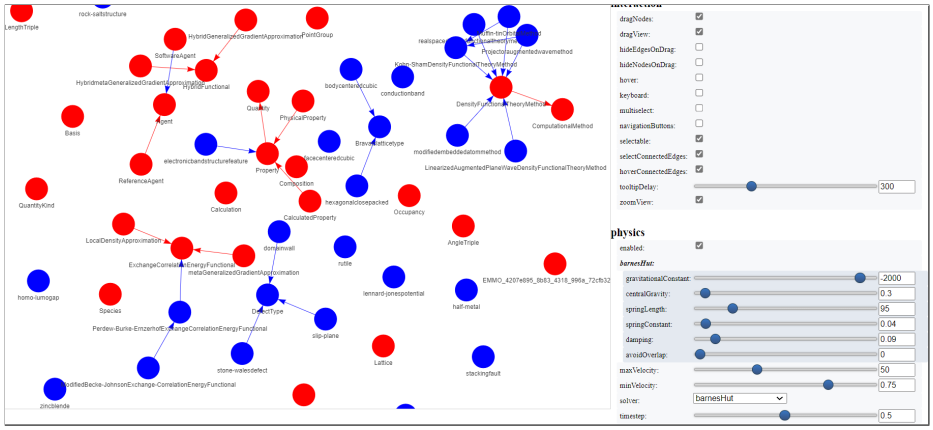


Fig. 10. Ontology visualization.

5.1 Use case - extending MDO

As a use case we decided to have a domain expert extend MDO using our tool. We used the list of 131 frequent phrases that was used in an earlier study described in [1] as well as the same domain expert.

The final result contains 29 new concepts (see Table 1) and 27 new axioms to be added to MDO.

The domain expert found that the tool gave support to the process of extending the ontology and was a large improvement over the Excel files that saved time and made the process easier. In particular, the fact that the tool guided the user through the different phases of the workflow was helpful. Further, the tool guides the user in different tasks and suggests possibilities. Also being able to search in the different lists and marking progress, as well as providing more information regarding the concepts and phrases, was very helpful. The labeling of the different kinds of changes was not always that easy and in the future we will remove this part for the end user. When we want to gather data about this, we will investigate in ways to do the labeling automatically.

When comparing the result with the result of [1], we noted some differences in the list of added concepts. The main reason for these differences was that the domain expert had a better understanding of the use of MDO and therefore sometimes made more restrictive decisions on what concepts should be part of MDO. Deciding on axioms is a task that the domain expert finds hard and it is clear that the help of an ontology engineering expert is highly recommended. Therefore, the current results regarding axioms should be considered as a first draft that should be discussed further.

5.2 User Study

As part of the design and implementation phase of our tool, we performed a small user study to receive feedback on the user interface and which resulted in some

Table 1. New concepts proposed to be added to MDO.

Stacking Fault	Linearized Augmented Plane Wave Density Functional Theory Method
Face Centered Cubic	Nearest Neighbor
Body Centered Cubic	Electronic Band Structure Feature
Rare Earth	Projector Augmented Wave Method
Brillouin Zone	Lennard-Jones Potential
Stone-Wales Defect	Rock-Salt Structure
Half-Metal	Realspace Density Functional Theory Method
Homo-Lumo Gap	Modified Becke-Johnson Exchange-Correlation Energy Functional
Valence Band	Perdew-Burke-Ernzerhof Exchange-Correlation Energy Functional
Slip-Plane	Kohn-Sham Density Functional Theory Method
Rutile	Hexagonal Close Packed
Conduction Band	Muffin-tin Orbital Method
Defect Type	Bravais Lattice Type
Zinc Blende	Modified Embedded Atom Method
Domain Wall	

changes to the user interface. The subjects were 4 PhD students in computer science at our department which did not have a background in ontologies nor in materials science. A small introduction including the purpose, the project and some basic ontology terminology was handed out beforehand. During a session with a subject via Zoom, the subject performed a number of typical tasks and was asked to think loud. The subjects did not think it was that easy to use the tool and we gathered a number of issues that led to changes that were implemented in the current version of the interface to improve the functionality of the tool. Some issues were left for a future version of the user interface. For instance, one issue that we will work on in the future is to make the interface more clean and not combine the functionality for an end user and a researcher gathering data about the use of the tool in one interface. Further, we will look into making the search functionality more easy to use and make other tasks more uniform.

6 Conclusion

In this paper we have presented the first version of a tool that supports users with phrase-based ontology extension by representing the user's workflow and guiding the user during each step. As future work, we will implement a version of the tool that is focused on end-users. We will also take the comments gathered from the use case and the user study into account. Additionally, it may be interesting to consider other kinds of relations besides *is-a* relation, as well as the fact that sometimes additions to the ontology may lead to deletions in the ontology as well.

Acknowledgements. This work has been financially supported by the Swedish e-Science Research Centre (SeRC), the Swedish National Graduate School in

Computer Science (CUGS), the Swedish Research Council (Vetenskapsrådet, dnr 2018-04147), and the Swedish Agency for Economic and Regional Growth (Tillväxtverket, 20201438).

References

1. Abd Nikooie Pour, M., Li, H., Armiento, R., Lambrix, P.: A first step towards extending the materials design ontology. In: ESWC Workshop on Domain Ontologies for Research Data Management in Industry Commons of Materials and Manufacturing (2021)
2. Asim, M.N., Wasim, M., Khan, M.U.G., Mahmood, W., Abbasi, H.M.: A survey of ontology learning techniques and applications. Database **2018**, bay101:1–24 (2018), doi: 10.1093/database/bay101
3. Buitelaar, P., Cimiano, P., Magnini, B.: Ontology Learning from Text: Methods, Evaluation and Applications. IOS Press (2005)
4. El-Kishky, A., Song, Y., Wang, C., Voss, C.R., Han, J.: Scalable topical phrase mining from text corpora. Proceedings of the VLDB Endowment **8**(3), 305–316 (2014), doi: 10.14778/2735508.2735519
5. Lambrix, P.: Completing and debugging ontologies: state of the art and challenges (2020), arXiv:1908.03171
6. Li, H., Armiento, R., Lambrix, P.: A method for extending ontologies with application to the materials science domain. Data Science Journal **18**(1) (2019), doi: 10.5334/dsj-2019-050
7. Li, H., Armiento, R., Lambrix, P.: An ontology for the materials design domain. In: Pan, J.Z., Tamma, V.A.M., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (eds.) The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12507, pp. 212–227. Springer (2020), doi: 10.1007/978-3-030-62466-8_14
8. Pesquita, C., Ivanova, V., Lohmann, S., Lambrix, P.: A framework to conduct and report on empirical user studies in semantic web contexts. In: Faron-Zucker, C., Ghidini, C., Napoli, A., Toussaint, Y. (eds.) Knowledge Engineering and Knowledge Management - 21st International Conference, EKAW 2018, Nancy, France, November 12-16, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11313, pp. 567–583. Springer (2018), doi: 10.1007/978-3-030-03667-6_36
9. Sabou, M., Fernandez, M.: Ontology (network) evaluation. In: Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.) Ontology engineering in a networked world, pp. 193–212. Springer (2012), doi: 10.1007/978-3-642-24794-1_9
10. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J., Groth, P., Goble, C., Grethe, J.S., Heringa, J., 't Hoen, P.A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR guiding principles for scientific data management and stewardship. Scientific data **3**, 160018:1–9 (2016), doi: 10.1038/sdata.2016.18

A Survey on User Interaction with Linked Data

Mariana Aguiar[1,2], Sérgio Nunes[1,2], and Bruno Giesteira[1,3]

[1] INESC TEC
[2] Faculty of Engineering, University of Porto, PORTUGAL
[3] Faculty of Fine Arts, University of Porto, PORTUGAL

Abstract. Since the beginning of the Semantic Web and the coining of the term Linked Data in 2006, more than one thousand datasets with over sixteen thousand links have been published to the Linked Open Data Cloud. This rising interest is fuelled by the benefits that semantically annotated and machine-readable information can have in many systems. Alongside this growth we also observe a rise in humans creating and consuming Linked Data, and the opportunity to study and develop guidelines for tackling the new user interaction problems that arise with it. To gather information on the current solutions for modelling user interaction for these applications, we conducted a study surveying the interaction techniques provided in the state of the art of Linked Data tools and applications developed for users with no experience with Semantic Web technologies. The 18 tools reviewed are described and compared according to the interaction features provided, techniques used for visualising one instance and a set of instances, search solutions implemented, and the evaluation methods used to evaluate the proposed interaction solutions. From this review, we can conclude that researchers have started to deviate from more traditional visualisation techniques, like graph visualisations, when developing for lay users. This shows a current effort in developing Semantic Web tools to be used by lay users and motivates the documentation and formalisation of the solutions encountered in the studied tools.

Keywords: Linked Data · Semantic Web · Visualisation · User Interaction.

1 Introduction

With the rise of interest in Linked Data in the recent years, we start to see more applications and platforms using this data modelling paradigm. This interest is fuelled by the benefits that semantically annotated and machine-readable information can have in many systems, like search engines and recommendation systems. However, most times Linked Data is still perceived as data that will only be created and consumed by machines and robots, and not by humans as well. As a result, guidelines and common practices for designing user interaction and to tackle the new problems that occur with developing Linked Data systems for lay users are lacking.

With this survey we aim to research and study the interaction solutions provided in the state of the art tools developed for inexperienced users to interact with Linked Data. This study is focused on works explicitly targeted at users with no previous knowledge of Semantic Web technologies. The goal is to gather information on the commonly used techniques for interacting with and visualising Linked Data for this subset of users. This study covers five types of interaction with content, analysing works that allow users to browse, search, visualise, create and edit Linked Data.

Section 2 presents related works that analyse and classify visualisation tools for Linked Data. Section 3 describes the methodology conducted to perform this literature review and the criteria followed for including or excluding works. Section 4, 5, and 6 will present a brief description about each work reviewed, by categorising them in full-featured tools, browsers and exploratory tools, and models and frameworks, respectively. Section 7 classifies the reviewed works according to their visualisation techniques, interaction actions, implemented search solutions, and conducted evaluation of the user interface. In Section 8, the main results reported in this study are described, and future work is outlined.

2 Related Work

Several works have been published aiming to study and analyse Linked Data visualisation tools. Some surveying all the visualisation tools found in the literature [3, 5, 23, 8], others focusing on more specific subsets of Linked Data tools like Linked Data consumption tools [18], ontology visualisation tools [17], and Linked Data exploration systems [20]. An overview of the works presented in this section can be consulted in Table 2.

In "Approaches to Visualising Linked Data: A Survey" [5] the authors start out by describing the key requirements that visualisations approaches for Linked Data must follow to be accessible to non-domain and non-technical users. It is defended that clear and coherent visualisation of Linked Data is essential to the use of the Semantic Web by everyone. In this survey the existing approaches to visualised Linked Data are reviewed and analysed according to the requirements for interactive visualisation outlined before and that the authors believe will lead to intuitive knowledge discovery for all users.

Similar to the previous work, in "Exploration and Visualization in the Web of Big Linked Data: A Survey of the State of the Art and Challenges Ahead" [3] the authors outline the prerequisites and challenges that should be followed by exploration and visualisation systems, and review the solutions developed by the Semantic Web community and analyse their alignment to the requirements. With this work the focus is the Web of Big Linked Data, and no requirements are outlined to ensure these systems are accessible and usable by users with no previous experience.

Another work surveying the state of the art of Linked Data visualisation tools is "Empirical evaluation of Linked Data Visualization Tools" [8]. In this work the authors formalised a set of goals that the tools must fulfil in order to "provide clear and convincing visualisations" and "encourage user comprehension". Each tool is then evaluated according to these goals to determine which ones can be used successfully by any Linked Data consumers.

"Interacting with Linked Data: A Survey from the SIGCHI Perspective" [23] is focused on studying the current interaction and visualisation approaches for Linked Data. This survey reviews Human-Computer Interaction (HCI) works on Linked Data and analyses them according to the following aspects: the end-users, the context, the design contributions, and how is the research validated. The authors highlight that there is little research supporting human end-users in querying, browsing, and visualising Linked Data, and that the ones that exist rarely include validation methods or theoretical conclusions.

In "Ontology Visualization Methods - A Survey" [17] a more specific set of Linked Data tools is analysed, specifically tools for visualising ontologies. A set of requirements is previously defined for this type of visualisation tools and the reviewed approaches were analysed according to it. The surveying tools were also categorised according to the different characteristics of the presentation, interaction techniques, functionalities supported, and visualisation dimensions. Another two surveys focused on smaller sets of Linked Data tools are a "Survey of Tools for Linked Data Consumption" [18] and "Survey of Linked Data based exploration systems" [20] where tools for Linked Data consumption and Linked Data exploration and search tools are reviewed, respectively.

Some of the previously mentioned works are focused on defining requirements that Linked Data visualisation tools must follow to be accessible to non-technical and lay users, and then reviewing all the state of the art approaches according to those requirements. However, none set their goal as surveying the state of the art of Linked Data visualisation tools and applications targeted at inexperienced users. With this survey on user interaction with Linked Data, we set out to analyse and review the Linked Data applications and visualisation tools developed with inexperienced users in mind to find their most commonly used visualisation techniques, interaction features, and search approaches. Instead of focusing on evaluating and determining which tools from the set of published solutions could be used by lay-users, we aim to find the most used visualisation and interaction techniques and the evaluation methods in the subset of tools explicitly designed for inexperienced users.

Table 1. Overview of the related works.

Work	Year	Description
Katifori et al. [17]	2007	Survey of ontology visualisation methods, and the evaluation of their features and characteristics according to a previously defined set of requirements.
Dadzie and Rowe [5]	2011	Survey of Linked Data visualisation tools, and their evaluation according to key requirements that tools must follow to be accesible by lay users.
Marie and Gandon [20]	2014	Survey of Linked Data based exploration systems.
Bikakis and Sellis [3]	2016	Survey of systems developed in the context of the Web of Linked Data, and their evaluation according to requirements for modern exploration and visualisation systems.
Klímek et al. [18]	2019	Survey of tools for Linked Data consumption, and their evaluation based on their usability by non-technical and non-domain experts.
Santo et al. [23]	2020	Survey of the Human-Computer Interaction contributions for the Linked Data research field.
Desimoni and Po [8]	2020	Survey of the current approaches for visualising and exploring Linked Data, and their evaluation according to a set of goals that allow tools to provide clear and comprehensible visualisations.

3 Methodology

Visualisation tools play a big part in studying user interaction with Linked Data. Even tough by definition visualisation tools are developed to offer its users features to explore the data to identify patterns, infer correlations, and causalities [3]; many tools featured in LD visualisation studies [21, 8] are in fact developed not only for people with previous experience with LD. The works featured in this literature review are tools and applications developed for inexperienced users and intended for navigating, searching, authoring, and visualising data, as opposed to analytical pattern and statistics extraction.

To conduct this literature review, we started out by defining 4 survey research questions by which we will guide our research. We aim to acquire knowledge about what visualisation techniques are commonly used to display Linked Data, what actions can the users perform in the applications, what are the most reoccurring search solutions, and information about the conducted evaluations. To answer these issues we defined the following survey research questions:

SRQ1: Which interaction actions can the user perform?
SRQ2: What visualisation techniques are commonly used?
SRQ3: Which search solutions are provided?
SRQ4: How is the user interface evaluated?

Next, we established inclusion and exclusion criteria to filter the works to be reviewed. We included works considered as LD visualisation tools, applications, or models to aid the development of user interfaces for LD applications; and works targeted at users with no previous knowledge of Semantic Web technologies. We excluded works not written in English and focused on tools for the extraction of statistics and analytical patterns from data.

Finally, we entered the following search query, `[("linked data" OR "semantic web") interaction]`, in several scientific databases, specifically Google Scholar[5], ResearchGate[6], and ACM Digital Library[7], and selected works according to the previously described criteria. Additionally, we collected works by recursively analysing for each article the citations from it and the citations to it.

In the end, 18 works were selected, which are presented in chronological order in the next three sections, each one corresponding to one category of works: Section 4 with full-featured tools, Section 5 with search and exploratory tools, Section 6 with models and frameworks. A list of the 18 works can be consulted in Table 2.

4 Full-featured tools

In this section we desribe the works that present platforms and applications for Linked Data that are focused on providing the user with a complete experience. These tools

[5] https://scholar.google.com
[6] https://www.researchgate.net
[7] https://dl.acm.org

provide ways for the users to explore and navigate the dataset, perform search queires, visualise individuals or groups of resources, and, in some cases create and edit data.

CS AKTiveSpace (CAS) [10] is a platform developed to take advantage of the semantic and distributed content collected related to Computer Science research in the United Kingdom. This application was developed, in part, to test the applicability to any domain of the mSpace model [9], described later in Section 6. Due to being based on mSpace, CAS user interface allows a semantic exploration of the domain tailored to its unique attributes. Users can explore the dataset by filtering down the instances by research area and location. An overview of the main instances, in this case researchers, is displayed to the user using a map, where they can select a specific area. When a researcher is selected, detailed information is also displayed in a simple textual format.

MuseumFinland [15] is a semantic portal for publishing heterogeneous museum collections in the Semantic Web. This application provides its users with several features, such as a global view of the collections, a content-based information retrieval system, and semantically linked and browsable content. Regarding the search features, MuseumFinland is equipped with a multi-faceted search system, that organises the resources in categories, as well as a simple keyword-based and geographical search. When selecting an instance in a result list, the user is redirected to a global view of that instance, composed of the instance image, metadata, and a set of hyperlinks to semantically related instances.

Ontowiki [1] is a Semantic Web tool developed for facilitating the visual representation and creation of a knowledge base for end users. This tool is focused on social collaboration aspects, and so it presents features such as change tracking, comments and discussions forums, rating and popularity metrics, and analysis of the users activity. To visualise a single instance, Ontowiki provides its users with an instance view that combines all of the properties and presents information about similar instances and incoming links. Regarding the search features provided, Ontowiki allows for facet-based browsing and a semantically enhanced full-text search. To author and edit the data, an inline edit mode is present in all information resulting from statements, following the "What You See Is What You Get" approach. The users can quickly edit or add statements by clicking a small edit or add button next to the information.

DBin [24] is a tool for contributing and exploring the Semantic Web by exchanging data in P2P channels. DBin is a tool that allows users to browse data of different domains, and each domain is equipped with predefined settings for visualising, querying and annotating the data with the help of "community configurable user interfaces", called Brainlets. A Brainlet defines which classes and relations between items are featured in DBin's tree layout navigator. This allows for the organisation of the information displayed to be targeted at the needs and common tasks of a certain domain. To aid inexperienced users to query the data, a Brainlet provides predefined common and useful queries to the domain, these are entitled as "precooked queries". Finally, to author and share the data, Brainlet lets its creator define custom domain dependent annotation templates, simple or more complex, and keep track of the annotations authors.

OKM Knowledge Management (OKM) [7] is a Linked Data authoring tool developed to analyse and test how lay users can interact with Linked Data systems. OKM is based on a traditional user interface with "resource-centric" pages, where each resource

has its own page with information and related resources. The users can follow these object properties to navigate the whole dataset. On top of this interface, the authors implemented some atypical features that were studied and analysed in order to assess their benefits to inexperienced users. One of the features was the inclusion of roles and templates. Instead of presenting a possibly exhaustive list of all the properties of a resource, the authors defend that a role must be defined for each class of the domain, where the most relevant information is displayed first. When adding a resource to the system the user must also assign one or more roles to it. Additionally, a template is provided so that all relevant information for that role is filled by the user. To evaluate the impact of these features, user experiments were conducted with participants with no previous knowledge of Linked Data.

The development of RDF Surveyor [26] was motivated by the lack of simple and intuitive solutions for user interfaces capable of being used by lay users to explore RDF data. The tool was developed to provide mechanisms to explore RDF datasets that hide the complex details of this kind of data. RDF Surveyor allows its users to navigate and explore the dataset, visualise an instance and search using a facet-based approach and a keyword-based approach. In the page of an instance the class types, data properties and related individuals are displayed, when applicable a picture and a map view are also provided.

5 Browsers and exploratory tools

Here we present browser and exploratory tools, meaning Linked Data tools and applications that are focused on providing the user with ways to navigate the datasets, or perform specific search tasks. Most tools presented next offer both navigation and search features, as well as, visualisation features.

QuizRDF [6] is a search engine proposed in 2004, to combine the advantages of free-text search with the advantages of browsing and searching for RDF metadata. The authors of this work defend that a typical user usually benefits more from starting with a single node or resource and exploring the remainder of the dataset from there; as opposed to starting with a general overview of the dataset. In order to do this, QuizRDF provides its users with an initial keyword search query, of ideally one or two terms, and a resulting ordered list of the results. Then, the users can further specify the search by selecting only the instances of a certain class, or the instances with a certain range of values for a given property. All the hyperlinks of properties relating the current class to others, or the super classes are clickable allowing the user to continue browsing and exploring the dataset.

Piggy Bank [14] is a browser extension developed to provide users with ways of using Semantic Web content while exploring and browsing the web. The main feature of this tool is that it allows users to extract information from web pages and transform it into RDF data. Piggy Bank also allows the user to visualise, browse and search for the data that they have saved. Upon saving a retrieved item, considered as a RDF resource that belongs to a class, has property values, and other retrieved resources, the user can assign tags and keywords to it. The items of each tag are then organised in collections, and faceted-search can be performed using them.

Tabulator [2] is a RDF browser developed for both inexperienced and experienced users to interact with the Semantic Web, and post and refine RDF data. Tabulator's main goal was to provide a generic Linked Data browser, that at the same time lets users benefit from user interface tools commonly encountered in domain specific applications. Similarly to other works, Tabulator allows the users to semantically browse the web of data, following the links between different resources. These resources are displayed in a tree view, where nodes can be expanded to obtain more information about them. Tabulator also allows the users to query for resources by providing values to different fields. The results can be visualised in different ways, with a table, timeline, map, or calendar view. In either mode, searching or exploring, when a resource is selected, the user is directed to a page with an outline view of the resource featuring more detailed information about it.

TreePlus [19] is a tool for visualising graphs that can be applied to several domains and different types of graphs, like the ones resulting from Linked Data datasets. This tool presents the graph to the users in a tree layout, allowing them to interactively explore it by starting with a single node and incrementally expanding it. The main goals while developing TreePlus were to keep the relationship labels readable and the tree layout as steady as possible. TreePlus offers its users features like a preview of the adjacent nodes, zooming and panning, simple keyword search and browsing and sorting options. The tests conducted concluded that TreePlus performed better in most of the tasks and that its benefits increase with the density of the graphs.

Ozone Browser [4] is a tool to make use of the semantics embedded in web documents in order to improve the experience of the users while browsing the web. The Ozone Browser is a graphical overlay that provides contextual knowledge of the hyperlinks in a web document. Regarding its user interface, Ozone Browser dynamically generates different views according to the domain of the RDF data presented. The default visualisation is composed of the RDF triples and related entities, if geographical data is available a map view is presented.

RelFinder [13] takes on a different approach than other browsing focused Linked Data tools. RelFinder's goal is to extract a graph that covers all the relationships between two predefined objects, and present it to the user. To this very simple graph visualisation, some interactive features were implemented to reduce the number of nodes and relations displayed. To start using this tool the user must select the two main objects using a basic keyword search, then the tool displays the graph connecting these two options. To filter the displayed relationships and nodes, users can select a subset according to their length or the classes they belong to, allowing them to better cater the visualisation to their interests and needs. Finally, if a node is selected in the graph, a simple visualisation of the instance information is displayed.

The approach for visual SPARQL queries based on filter/flow graphs [11] was developed for users with no knowledge or experience with RDF data to be able to create complex queries and obtain in-depth knowledge. This approach is composed of filters on the data or object properties of the data, that are connected as a graph. These filters include restrictions on ordinal values and strings, or resource identifiers. This tool underwent a qualitative user evaluation to assess the users ability to comprehend and compose SPARQL queries. While the results show that most users were able to quickly

complete the tasks, half of the participants already had previous basic knowledge of Semantic Web technologies. With these results we cannot be sure the tool is suited for users with no experience with Linked Data.

PepeSearch [25] is a Linked Data tool for searching semantic datasets, targeted at mainstream users, and thus requires no understanding or previous experience with Semantic Web data formats. PepeSearch provides its users with searching and instance visualisation features. Similar to other works, Pepesearch also allows its users to use a multi-faceted search, where they can apply different constraints to the data and object properties of all the classes in the dataset. The results are presented in a table with the values of all properties, and when an instance is selected the user is redirected to a page dedicated to that instance. In this page the data and object properties of the instance are displayed and the user can follow the links to the pages of the related instances.

6 Models and Frameworks

In this section, we feature works that present formalised interaction models created for, or adapted to Linked Data environments. Additionally, we present software frameworks to develop user interfaces and model user interaction for semantic platforms.

The mSpace model [9] is an interaction model that can be used for interactions with Semantic Web applications. This model provides developers and designers with an intuitive and effective user interface that accommodates alternatives to the commonly used keyword search. Initially, the mSpace model was not developed for Semantic Web applications, but motivated by the lack of research in user interaction with Linked Data, the authors decided on the formalisation of mSpace as an interaction model for the Semantic Web. mSpace is focused on facilitating the interaction with Semantic Web data to perform user-determined exploration of a certain domain. The default visualisation of this model is a multi-column table view, where data classes are ordered from left to right. The users can then browse through the data selecting an instance in each column. By selecting sequentially one instance from each column, the instances are filtered according to their relations, this browsing mechanism allows the users to chose their path while navigating the domain. Additionally, to better accommodate the user interests, the classes presented in the table can be rearranged, or substituted by others. To complement the browsing approach, mSpace provides users with a detailed view for the last selected instance, that displays contextual information about it.

Haystack [22] is a platform to facilitate the development of Linked Data applications, that allow lay users to create, manipulate, and visualise RDF data. Haystack focuses on providing developers and designers with user interface components that were designed specially to handle the heterogeneity of RDF data and contribute to a good user experience. Haystack allows designers to specify different types of views, differing in size and location, for every type of resource in the domain. Additionally, developers can also add customisable operations tailored to a resource, available in the view displayed to the user. This tool provides users with a keyword-based search, a simple textual visualisation of each resource, and mechanisms to author, edit, and annotate the resources. Finally, users can semantically explore the domain by following the links of the related resources.

Table 2. Overview of the interaction actions provided by the works reviewed.

	Exploration/ Navigation	Search	Author/ Edit	Visualisation
Full-featured tools				
CS AKTiveSpace (2004) [10]	✓			✓
MuseumFinland (2005) [15]	✓	✓		✓
Ontowiki (2006) [1]		✓	✓	✓
DBin (2006) [24]	✓	✓	✓	✓
OKM (2010) [7]	✓	✓	✓	✓
RDF Surveyor (2019) [26]	✓	✓		✓
Search and exploratory tools				
QuizRDF (2004) [6]	✓	✓		✓
Piggy Bank (2005) [14]	✓	✓		✓
Tabulator (2006) [2]	✓	✓		✓
TreePlus (2006) [19]	✓	✓		
Ozone Browser (2009) [4]	✓			✓
RelFinder (2009) [13]	✓	✓		✓
Filter/Flow (2014) [11]		✓		
PepeSearch (2016) [25]		✓		✓
Models and frameworks				
mSpace (2003) [9]	✓			✓
Haystack (2003) [22]	✓	✓	✓	✓
Information Workbench (2011) [12]	✓	✓	✓	✓
Sampo-UI (2020) [16]	✓	✓		✓

The Information Workbench [12] is a platform that supports the whole development process of developing a Linked Data application. Here, we will be focused on reviewing the options and chosen solutions for the user interface design. Firstly, the platform supports the implementation of basic a keyword search, as well as graph pattern search and faceted search. The Information Workbench provides the developers with a wide range of different visualisation widgets to accommodate the highly heterogeneity of Linked Data datasets. For a set of instances or resources views like maps, lists, timelines, and graphs are provided. For a single instance information can be consulted in a generic textual view, tabular view, or graph view, where only the neighbourhood of the instance selected is displayed. In all these visualisation options the user can semantically browse the dataset by clicking in the related instances hyperlinks. Additionally to searching, visualising and browsing the data, users are provided with simple widgets to edit and annotate the presented data.

Sampo-UI [16] is a software framework for developing customisable and responsive user interfaces for semantic portals. The authors started out by outlining the requirements for the user interfaces for Semantic Web applications, based on their experience in developing these semantic portals. The first requirement presented is providing the user with a variety of different ways of exploring the data, for this the authors highlight some search paradigms: free-text search, faceted search, geospatial search, and temporal search. Another requirement is providing the users with different views of the search results, in the form of tables, lists, geospatial, or temporal visualisations. The final requirement outlined is that the user interface should support semantic browsing, meaning semantic links should allow the users to browse through the dataset, from one instance's page to another. These instance pages are described as a landing page for an

[7] http://conference-explorer.fluidops.net/

Table 3. Overview of the visualisation techniques used in the reviewed works.

	Instance View					Instance Set View				
	Textual	Map	Calendar	Table	Graph	List	Table	Tree	Map	Timeline	Graph
Full-featured tools											
CS AKTiveSpace (2004) [10]	✓					✓	✓		✓		
MuseumFinland (2005) [15]	✓					✓					
Ontowiki (2006) [1]	✓	✓	✓			✓	✓				
DBin (2006) [24]	✓							✓			
OKM (2010) [7]	✓					✓					
RDF Surveyor (2019) [26]	✓	✓				✓					
Search and exploratory tools											
QuizRDF (2004) [6]	✓					✓					
Piggy Bank (2005) [14]	✓					✓	✓				
Tabulator (2006) [2]	✓					✓	✓	✓	✓	✓	
TreePlus (2006) [19]								✓			
Ozone Browser (2009) [4]	✓	✓									
RelFinder (2009) [13]	✓					✓	✓		✓	✓	✓
Filter/Flow (2014) [11]	✓			✓							
PepeSearch (2016) [25]	✓					✓					
Models and frameworks											
mSpace (2003) [9]	✓						✓				
Haystack (2003) [22]	✓					✓					
Information Workbench (2011) [12]	✓			✓	✓	✓	✓		✓	✓	✓
Sampo-UI (2020) [16]	✓					✓	✓		✓	✓	

entity, providing information about its metadata, and internal and external links to other related entities. The development of Sampo-UI was based on these requirements and components to implement these features were developed.

Table 4. Overview of the evaluation methods conducted.

	None	Qualitative	Quantitative
Full-featured tools			
OKM (2010) [7]		Satisfaction Study (71)	User Study (71)
RDF Surveyor (2019) [26]			Usability Study (14)
Search and exploratory tools			
TreePlus (2006) [19]		User Study (28)	
Filter/Flow (2014) [11]		User Study (10)	
PepeSearch (2016) [25]			Usability Study (15)

7 Discussion

To answer the survey research questions previously stated, each reviewed work was classified according to its interaction actions, visualisation techniques, search solutions and user interface evaluation. The information gathered was then systematised in 4 tables and analysed. Regarding SRQ1 we categorised the works according to the main interaction actions provided to the users, summarised in Table 2. These were divided

in 4 categories: (1) exploration/navigation, meaning the users were able to semantically browse the data by following related resources, and without the need to always start a new search task; (2) search, where users can query the system for a single or a group of resources; (3) author/edit, where the users are provided with mechanisms to edit, create, or annotate resources; (4) visualisation, applications where users have access to the visualisation of a single resource or a group of related resources. We can conclude that the majority of works provide ways for users to explore, search and visualise Linked Data, with only a small group of works providing features to author and edit. We can assume the lack of tools for authoring data is due to the fact that most targeted at lay-users are focused on browsing and visualising it, as opposed to focused on users contributing to the dataset. Another reason might be the complexity of designing an interface that motivates the users to input semantically correct RDF data, with no previously knowledge of these technologies. It is also worth mentioning that the two works with no visualisation features [19, 11] are solely focused on the development of a novel approach for lay-users to construct SPARQL queries and exploring graph-based data, respectively.

Table 5. Overview of the search techniques in the reviewed works.

	Facet-based	Keyword-based	Predefined queries	Geographical	Temporal
Full-featured tools					
MuseumFinland (2005) [15]	✓	✓		✓	
Ontowiki (2006) [1]	✓	✓			
DBin (2006) [24]			✓		
OKM (2010) [7]	✓				
RDF Surveyor (2019) [26]	✓	✓			
Search and exploratory tools					
QuizRDF (2004) [6]		✓			
Piggy Bank (2005) [14]	✓				
Tabulato (2006)r [2]			✓		
TreePlus (2006) [19]		✓			
RelFinder (2009) [13]		✓			
Filter/Flow (2014) [11]	✓				
PepeSearch (2016) [25]	✓				
Models and frameworks					
mSpace (2003) [9]	✓				
Haystack (2003) [22]	✓				
Information Workbench (2011) [12]	✓	✓			
Sampo-UI (2020) [16]	✓	✓		✓	✓

To survey the visualisation techniques in the reviewed works, we decided to analyse two different kinds of visualisation, the visualisation of a single instance, and the visualisation of a set of instances. The gathered information to answer SRQ2 is summarised in Table 3. For the single instance visualisation, the most common technique is a textual view of the resource, where the related information, data properties, object properties, and other related resources are listed. However, several works highlight the fact that this approach can quickly make the interface too cluttered and dense, when the dataset used is highly connected and the classes have a large number of properties. To combat this we encountered some approaches using collapsible sections, to hide and show more information upon a user's need. The map and table techniques consist of only dis-

playing the related resources with geographical and date data, respectively. The graph view for a single instance, only used in the Information Workbench model [12], is used to display the neighborhood of related resources, suitable to visualise social networks or family trees. Regarding the visualisation techniques for a set of instances, the most common approaches are lists and tables, the only difference between the two being that a table usually displays some of the instance's attributes, while a list only displays the instance's label. Surprisingly, the least common approach was the graph visualisation, represented in Figure 1, usually deeply connected to the visualisation of Linked Data, due to its natural network format. One explanation for this observation is that all tools are developed to be used by anyone with no previous experience with Linked Data, and graph visualisations, specially when large and highly connected, usually result in usability issues for this kind of users.

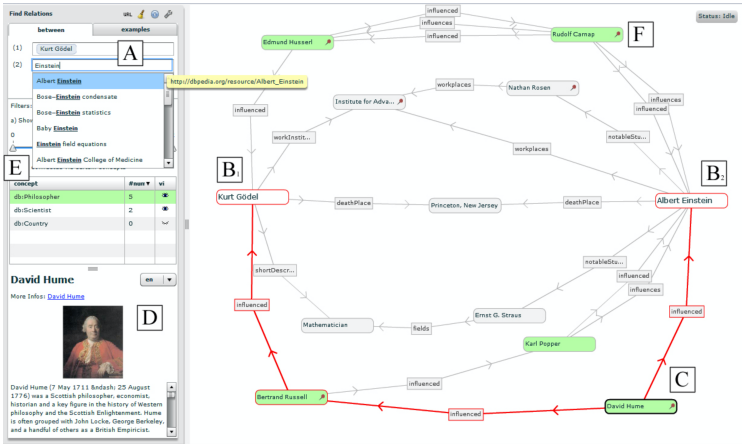


Fig. 1. Example of a graph visualisation in RelFinder [13]

To answer SRQ3 we collected the search solutions available in the subset of works that provided users with features to search for a single or a set of resources. Table 5 summarises the gathered information, and clearly shows that the two most common approaches are faceted-based search and keyword-based search. Tools that offer a faceted-based search allow users to search for resources specifying the class they belong to, data property values, or other related resources. We believe a faceted-based search is the most promising solution for semantic datasets, as it allows users to benefit from the semantic knowledge of the data. However, we find it best used together with the more traditional keyword-based approach, where users can search for resources by matching the entered keywords with the resources' label or other literal attributes.

Finally regarding SRQ4, we categorised the works according to the type of evaluation conducted to validate the user interface, summarised in Table 4. A large majority of around 70% of the works reviewed did not formally test their interfaces, which proves the lack of research and evaluation in user interaction with Linked Data systems.

8 Conclusions and Future Work

In this paper, we reviewed the state of the art Linked Data tools and applications developed for inexperienced users. Our goal was to gather knowledge on common approaches and solutions for user interaction design with Linked Data by analysing which user interaction features, like visualising techniques and search solutions, are more commonly provided in these tools. We can point out some recurring solutions, such as the use of faceted search interfaces and several data visualisation options when consulting a single or a group of resources. Surprisingly, graph-based visualisations, traditionally associated with Linked Data, were the least common approach encountered in the set of works reviewed. Usually, some of these more traditional visualisation and interaction techniques come with usability issues and are not familiar to users with no previous knowledge of Linked Data technologies, especially when associated with large and highly interconnected datasets. While these traditional approaches may negatively impact the use of the web of data by inexperienced users, the alternative visualisation techniques and user interaction approaches that have been developed and studied in the literature show that researchers have started to consider Linked Data as a data modelling paradigm also to be consumed by humans. We outline the future work for this study as the systematic documentation and abstraction of the alternative solutions found with this research, as we consider them necessary to broaden the use of Linked Data and the Semantic Web by lay-users.

Acknowledgements

This work was supported by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia within project EPISA DSAIPA/DS/0023/2018.

References

1. Auer, S., Dietzold, S., Riechert, T.: Ontowiki - A tool for social, semantic collaboration. In: Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4273, pp. 736–749. Springer (2006). https://doi.org/10.1007/11926078_53, https://doi.org/10.1007/11926078_53
2. Berners-Lee, T., Chen, Y., Chilton, L., Connoll, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and analyzing linked data on the semantic web. In: Proceedings of the 3rd International Semantic Web User Interaction (2006), http://swui.semanticweb.org/swui06/papers/Berners-Lee/Berners-Lee.pdf
3. Bikakis, N., Sellis, T.K.: Exploration and visualization in the web of big linked data: A survey of the state of the art. In: Palpanas, T., Stefanidis, K. (eds.) Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016. CEUR Workshop Proceedings, vol. 1558. CEUR-WS.org (2016), http://ceur-ws.org/Vol-1558/paper28.pdf
4. Burel, G., Cano, A.E., Lanfranchi, V.: Ozone browser: Augmenting the web with semantic overlays. In: Auer, S., Bizer, C., Grimnes, G.A. (eds.) Proceedings of the Fifth Workshop on

Scripting and Development for the Semantic Web , co-located with 6th European Semantic Web Conference, SFSW@ESWC 2009, Heraklion, Greece, May 31, 2009. CEUR Workshop Proceedings, vol. 449. CEUR-WS.org (2009), http://ceur-ws.org/Vol-449/Challenge1.pdf

5. Dadzie, A., Rowe, M.: Approaches to visualising linked data: A survey. Semantic Web **2**(2), 89–124 (2011). https://doi.org/10.3233/SW-2011-0037, https://doi.org/10.3233/SW-2011-0037
6. Davies, J., Weeks, R.: Quizrdf: Search technology for the semantic web. In: 37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM / Abstracts Proceedings, 5-8 January 2004, Big Island, HI, USA. IEEE Computer Society (2004). https://doi.org/10.1109/HICSS.2004.1265293, https://doi.org/10.1109/HICSS.2004.1265293
7. Davies, S., Hatfield, J., Donaher, C., Zeitz, J.: User interface design considerations for linked data authoring environments. In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M. (eds.) Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010. CEUR Workshop Proceedings, vol. 628. CEUR-WS.org (2010), http://ceur-ws.org/Vol-628/ldow2010_paper17.pdf
8. Desimoni, F., Po, L.: Empirical evaluation of linked data visualization tools. Future Gener. Comput. Syst. **112**, 258–282 (2020). https://doi.org/10.1016/j.future.2020.05.038, https://doi.org/10.1016/j.future.2020.05.038
9. Gibbins, N., Harris, S., Schraefel, M.: Applying mspace interfaces to the semantic web. In: World Wide Web Conference 2004 (01/01/04) (2003), https://eprints.soton.ac.uk/258639/
10. Glaser, H., Alani, H., Carr, L., Chapman, S., Ciravegna, F., Dingli, A., Gibbins, N., Harris, S., Schraefel, M.M.C., Shadbolt, N.: CS aktivespace: Building a semantic web application. In: Bussler, C., Davies, J., Fensel, D., Studer, R. (eds.) The Semantic Web: Research and Applications, First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece, May 10-12, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3053, pp. 417–432. Springer (2004). https://doi.org/10.1007/978-3-540-25956-5_29, https://doi.org/10.1007/978-3-540-25956-5_29
11. Haag, F., Lohmann, S., Bold, S., Ertl, T.: Visual SPARQL querying based on extended filter/flow graphs. In: Paolini, P., Garzotto, F. (eds.) International Working Conference on Advanced Visual Interfaces, AVI 2014, Como, Italy, May 27-29, 2014. pp. 305–312. ACM (2014). https://doi.org/10.1145/2598153.2598185, https://doi.org/10.1145/2598153.2598185
12. Haase, P., Schmidt, M., Schwarte, A.: The information workbench as a self-service platform for linked data applications. In: Hartig, O., Harth, A., Sequeda, J.F. (eds.) Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011. CEUR Workshop Proceedings, vol. 782. CEUR-WS.org (2011), http://ceur-ws.org/Vol-782/HaaseEtAl_COLD2011.pdf
13. Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., Stegemann, T.: Relfinder: Revealing relationships in RDF knowledge bases. In: Chua, T., Kompatsiaris, Y., Mérialdo, B., Haas, W., Thallinger, G., Bailer, W. (eds.) Semantic Multimedia, 4th International Conference on Semantic and Digital Media Technologies, SAMT 2009, Graz, Austria, December 2-4, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5887, pp. 182–187. Springer (2009). https://doi.org/10.1007/978-3-642-10543-2_21, https://doi.org/10.1007/978-3-642-10543-2_21
14. Huynh, D., Mazzocchi, S., Karger, D.R.: Piggy bank: Experience the semantic web inside your web browser. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings. Lecture Notes in Computer

Science, vol. 3729, pp. 413–430. Springer (2005). https://doi.org/10.1007/11574620_31, https://doi.org/10.1007/11574620_31
15. Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M., Kettula, S.: Museumfinland - finnish museums on the semantic web. J. Web Semant. **3**(2-3), 224–241 (2005). https://doi.org/10.1016/j.websem.2005.05.008, https://doi.org/10.1016/j.websem.2005.05.008
16. Ikkala, E., Hyvönen, E., Rantala, H., Koho, M.: Sampo-ui: A full stack javascript framework for developing semantic portal user interfaces. Semantic Web–Interoperability, Usability, Applicability (2020)
17. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.G.: Ontology visualization methods - a survey. ACM Comput. Surv. **39**(4), 10 (2007). https://doi.org/10.1145/1287620.1287621, https://doi.org/10.1145/1287620.1287621
18. Klímek, J., Skoda, P., Necaský, M.: Survey of tools for linked data consumption. Semantic Web **10**(4), 665–720 (2019). https://doi.org/10.3233/SW-180316, https://doi.org/10.3233/SW-180316
19. Lee, B., Parr, C.S., Plaisant, C., Bederson, B.B., Veksler, V.D., Gray, W.D., Kotfila, C.: Treeplus: Interactive exploration of networks with enhanced tree layouts. IEEE Trans. Vis. Comput. Graph. **12**(6), 1414–1426 (2006). https://doi.org/10.1109/TVCG.2006.106, https://doi.org/10.1109/TVCG.2006.106
20. Marie, N., Gandon, F.L.: Survey of linked data based exploration systems. In: Thakker, D., Schwabe, D., Kozaki, K., García, R., Dijkshoorn, C., Mizoguchi, R. (eds.) Proceedings of the 3rd International Workshop on Intelligent Exploration of Semantic Data (IESD 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 20, 2014. CEUR Workshop Proceedings, vol. 1279. CEUR-WS.org (2014), http://ceur-ws.org/Vol-1279/iesd14_8.pdf
21. Po, L., Bikakis, N., Desimoni, F., Papastefanatos, G.: Linked Data Visualization: Techniques, Tools, and Big Data. Morgan & Claypool Publishers LLC (March 2020). https://doi.org/10.2200/s00967ed1v01y201911wbe019
22. Quan, D., Huynh, D., Karger, D.R.: Haystack: A platform for authoring end user semantic web applications. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2870, pp. 738–753. Springer (2003). https://doi.org/10.1007/978-3-540-39718-2_47, https://doi.org/10.1007/978-3-540-39718-2_47
23. Santo, A.D., Holzer, A.: Interacting with linked data: A survey from the SIGCHI perspective. In: Bernhaupt, R., Mueller, F.F., Verweij, D., Andres, J., McGrenere, J., Cockburn, A., Avellino, I., Goguey, A., Bjøn, P., Zhao, S., Samson, B.P., Kocielnik, R. (eds.) Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI 2020, Honolulu, HI, USA, April 25-30, 2020. pp. 1–12. ACM (2020). https://doi.org/10.1145/3334480.3382909, https://doi.org/10.1145/3334480.3382909
24. Tummarello, G., Morbidoni, C., Nucci, M.: Enabling semantic web communities with dbin: An overview. In: Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4273, pp. 943–950. Springer (2006). https://doi.org/10.1007/11926078_69, https://doi.org/10.1007/11926078_69
25. Vega-Gorgojo, G., Giese, M., Heggestøyl, S., Soylu, A., Waaler, A.: Pepesearch: Semantic data for the masses. PLOS ONE **11**(3), 1–12 (03 2016). https://doi.org/10.1371/journal.pone.0151573, https://doi.org/10.1371/journal.pone.0151573

26. Vega-Gorgojo, G., Slaughter, L., von Zernichow, B.M., Nikolov, N., Roman, D.: Linked data exploration with RDF surveyor. IEEE Access **7**, 172199–172213 (2019). https://doi.org/10.1109/ACCESS.2019.2956345, https://doi.org/10.1109/ACCESS.2019.2956345

Visual Presentation of SPARQL Queries in ViziQuer *

Kārlis Čerāns, Jūlija Ovčiņņikova, Mikus Grasmanis, Lelde Lāce, and Aiga Romāne

Institute of Mathematics and Computer Science, University of Latvia
karlis.cerans@lumii.lv

Abstract. Visual presentation of information artefacts can ease their perception. There are several solutions allowing visual-centered composition of SPARQL queries over RDF data endpoints by an end-user. We describe a "reverse" method and tool for generating visualizations in a UML-style diagram notation for a rich set of existing SPARQL queries (including BGPs, value filters, optional and negated constructs, as well as unions, aggregation, grouping and subqueries). Such a visualization is expected to enhance the reading, understanding, sharing and re-use ability for SPARQL queries, as well as it can be helpful in learning SPARQL itself. The prototype for visual notation generation from a SPARQL query is implemented within the ViziQuer tool environment. We describe and demonstrate the SPARQL query visualization possibilities on example query sets over Europeana cultural heritage data and other SPARQL endpoints.

Keywords: RDF · SPARQL · Visual queries · Query visualization · ViziQuer

1 Introduction

Visual presentation of information artefacts can ease their perception. Visual query composition paradigm (cf. [2, 8, 9, 12, 14]) along with facet-based ([10, 13]) and controlled natural language based (cf. [7]) approaches offers a promising avenue for involving end-users in query composition over SPARQL endpoints (cf. [12]). Although most of the visual query composition tools support visual composition of just the simplest forms of conjunctive SPARQL queries, with OptiqueVQs [12] and LinDA [9] supporting also outer-level aggregation, the ViziQuer notation and environment (cf. [2, 4, 5]) provides visual means for rich query definition, involving BGPs, value filters, optional and negated constructs, as well as unions, aggregation, grouping and subqueries, coming close to the full SPARQL 1.1 SELECT query visualization [2].

In this paper we present a "reverse" method for creating a visual presentation of an existing SPARQL query that complements the visual query creation process e.g., by offering an option to start the query creation from an already existing SPARQL form. It can help understanding a textual SPARQL query, therefore facilitating query sharing, communication, and re-use, as well as learning and adopting the visual query notation and eventually also the SPARQL notation itself. Our proposal is to use the ViziQuer notation [2, 4] as the SPARQL query visualization target, as it is sufficiently rich to

cover a wide range of query constructs, and it has an execution environment[1] [5] that can translate the visual query back into SPARQL and execute it over the data endpoint.

Automated visualization of (rich) SPARQL queries can be expected to facilitate the existing query comprehension. It would also allow combining the visual query creation approach with textual query writing and possibly also with other end-user-oriented query writing approaches, as e.g., SPARKLIS [7] that supports query creation in controlled natural language, to obtain integrated multi-paradigm data query environments.

We show in this paper that a SPARQL query can be rendered in a visual UML class diagram style notation, taking advantage of presenting a query variable or URI together with its class assertion and presenting its properties in attribute or link notation; such work has not been done before for rich sets of SPARQL queries.

Visualizing of textual queries is common within the realm of relational databases, where visual query builders for most DB management systems can be initialized by a textual SQL query and the visual and textual query presentations are kept in synchronization. There are efforts regarding SPARQL query visualization [11] in Optique VQs context, yet these consider only the simple query forms covered by [12].

We presume that adding a compact visual presentation to a SPARQL query can contribute to easing its perception (if comparing textual SPARQL query together with the visual presentation to the textual SPARQL query alone) if the visual query presentation is of similar or lower structural complexity than the original textual query. The target group of the query visualization would include both the "lay users" that would be able to obtain a structural presentation of the SPARQL query, as well as data management professionals that would be able to use the visual query presentation together with the textual one (in SPARQL).

Within the rest of the paper, Section 2 reviews the ViziQuer visual query notation, Section 3 describes the principles of mapping the SPARQL constructs onto the visual ones, Section 4 presents and discusses the SPARQL visualization evaluation, Section 5 describes the related work and Section 6 concludes the paper. The resources supporting the paper are available from http://viziquer.lumii.lv/examples/sparql2viziquer.

2 Visual UML Style Query Notation Overview

We start with a review of the visual RDF data query notation, as available in ViziQuer [2, 4]. The visual queries are formulated in the context of a data dictionary that is a part of the data schema used by the ViziQuer tool and maps the entity short names used in the visual query presentations to their full IRIs.

Table 1 lists a few visual query examples over the Europeana cultural heritage data SPARQL endpoint. Further visual query examples are in [2, 3] and the tool web-site. The principal elements within a query are its ***nodes*** and ***edges***, the ***main node*** in a query is depicted as an orange round rectangle. A node in a query is either a ***data node*** that describes a variable or an IRI to be used as a subject or object within a triple pattern in a SPARQL query, or a ***control node*** use to further shape the query structure, if necessary (cf. the [ ] node in Example 4 in Table 1). The variable name or IRI can be explicitly specified for a data node, or the variable name can be implicit.

[1] cf. http://viziquer.lumii.lv

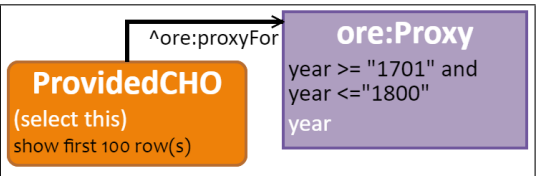
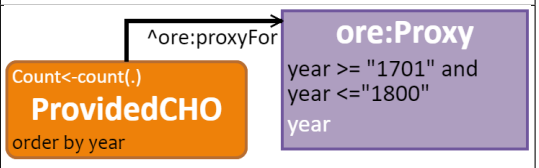
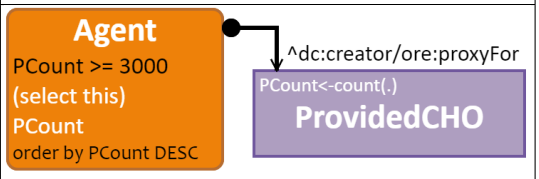
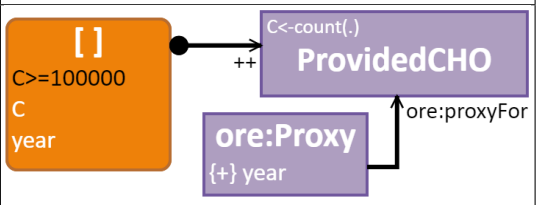
1. *List top 100 provided cultural heritage objects from the 18th century.* Each query node is a data instance pattern, listing its class name and conditions, and the selection items; the links show the instance connections.	
2 *Count the provided cultural heritage objects for each year during the 18th century*. The grouping is automatic by all non-aggregated selection fields.	
3. *Find all agents that are creators of at least 3000 provided cultural heritage objects.* The subquery construct (on *^dc:creator/ore:proxyFor* link) is used for calculating the CHO count for every agent separately.	
4. *Find all years corresponding to more than 100000 cultural heritage objects.* The subquery (over *ProvidedCHO* and *ore:Proxy*) computes the CHO count for each year. The outer query with unit node [ ] filters the subquery results.	

Table 1: Visual Query Examples

The following optional information can be specified for each data node:

- A ***class name*** (encoding the *rdf:type* assertion of the node variable of IRI),
- An ordered list of ***attribute fields***; each field specifies a variable in the query, together with its relation to the node variable or IRI, either via a Basic Graph pattern (BGP) triple, or as a BIND-expression; there are markers {+} for a required attribute value and {h} for not including the variable into the query select list (such a variable remains a helper for other value and condition description),
- A list of ***conditions*** (filters), each a Boolean-valued expression to be evaluated in the context of the node (shown in dark font), and
- A list of additional ***grouping markers*** (including a dedicated option to mark the node variable itself as grouping); this applies to aggregated queries.

The ***principal*** (bold) ***edges*** of a query determine its ***tree-shape structure*** with the main query node being its root. An edge can be labelled by a property path denoting its end node variable/URI connection, or it can be marked by == denoting the same variable on both ends, or ++ standing for no specified data connection between the edge ends; further connections between nodes can be specified by explicit references to the node and field aliases. There can be also ***auxiliary edges*** besides the query principal edge tree-shape structure to specify the extra data connections visually. An edge can also

carry an ***explicit variable*** in the form *?p* for a SPARQL triple with a variable in the predicate position (use the form *??p* if the variable is not included in the selection set).

A principal edge can correspond to a ***join link***, or an edge can link a host node to a ***subquery*** (starting with a black dot, as in Table 1, examples 3 and 4); the scope of a subquery involves the entire graph fragment beyond the subquery link, as well as the link itself and a reference to the variable or IRI in the host node (the node just above the subquery link). An edge can be ***required***, ***optional*** or ***negated***. Auxiliary edges always are join links; it does not make sense to have an auxiliary edge optional.

The main query node, as well as the head nodes of subqueries can have the following information (that applies to the entire main query or subquery level), as well:

- A list of ***aggregate fields*** (each with an aggregate function, aggregation expression, and a distinct element marker); these are positioned above the node class name,
- A ***distinct values*** specification to apply the DISTINCT modifier to the solution set (not to be used together with aggregate fields)
- An ordered list of ***result order modifiers*** (ORDER BY)
- Result ***slicing modifiers*** LIMIT and OFFSET (main query node only, or a subquery marked with an additional 'global' option).

There are two control nodes available in ViziQuer: unit [ ] and union [+]. A unit node can serve as a non-data bound container for outer-level processing of data (e.g., filtering or further aggregating) returned by a single or several subqueries. A union node corresponds to the union of the data sets corresponding to its branches (a branch can be linked to the union either by a join, or by a subquery link).

The visual query notation is explained in further detail and translation of a visual query into SPARQL are described in [4].

3 SPARQL Query Visualization

The visualization of a SPARQL SELECT query produces a visual extended UML-style diagram that describes the entire query contents. For a simple query consisting of the graph patterns, the pattern subject and object variables and resources are depicted as the query graph nodes and the graph edges correspond to their connections, with the important optimizations to represent the variable/resource ***classes*** in the dedicated class name compartments and ***single-use objects*** of SPARQL triples within node attribute fields, so obtaining a compact query presentation. Figure 1 shows an example query adopted from [1] in SPARQL form, and automatically rendered in the visual notation (with manual tuning of the node placement); the common namespace prefix definition in the SPARQL query is omitted for brevity.

The nodes in the visual query correspond to the variables *?ProvidedCHO*, *?Aggregation* and *?Proxy*. The variable *?year* is rendered into the visual notation in the attribute form. There are no class names in the visual query, as the original query contains no class specification. The SPARQL query visualization algorithm can create also visual query forms (up to the placement of the nodes and edges) from the SPARQL query texts for all examples, shown in Table 1. In the case of a query consisting of several nested blocks, the visual representation is first created for each of the blocks separately,

followed by merging the block representations into a single visual query. The rendering of expressions is generally carried over from the SPARQL textual notation into the textual compartments of the visual query notation.

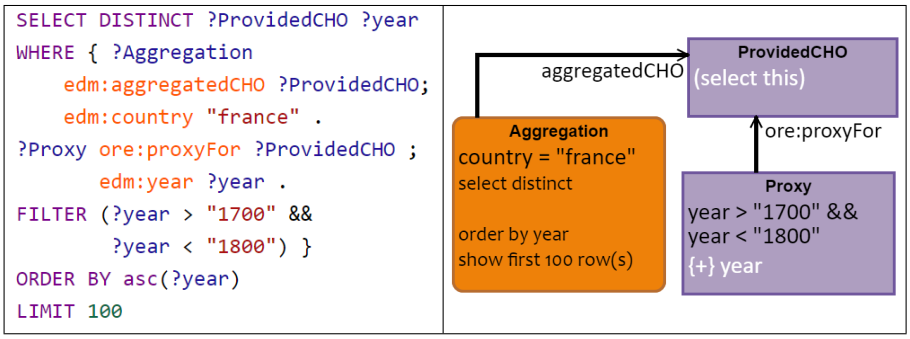


Fig. 1: Objects provided to Europeana from the 18th century from France

In what follows, we outline the SPARQL SELECT query visualization algorithm in more detail. The current version of the algorithm assumes that the SPARQL query to be visualized conforms to the following restrictions:

1. there are no HAVING clauses in the query[2],
2. the query does not use the FROM, GRAPH and SERVICE clauses, and
3. the EXISTS/NOT EXISTS clauses in the FILTER expressions are on the top expression level (i.e., they are not within other logical expressions).

The queries not satisfying 1 can be re-written into an equivalent form, satisfying the condition. The issue 2 is a limitation induced by the current state of the visual notation not providing the support of the respective constructs. The issue 3 can be resolved by resorting to richer textual expression fields within the visual query frames; it has not been considered at the current point.

The translation of a SPARQL query into the UML-based ViziQuer visual notation is based on the inclusion hierarchy of the following ***blocks*** within the query:

1. the query itself and its subqueries, and
2. the ***inline blocks*** that are group graph patterns behind group-or-union construct (simple parenthesis { } or UNION clause), OPTIONAL[3], MINUS, or EXISTS[4]/NOT EXISTS fragments within FILTER clauses.

[2] There is no direct counterpart for the HAVING clauses in the visual notation currently, as it is preferred to model their effects by means of subqueries

[3] An OPTIONAL block that consists of a single triple is considered to be a part of the enclosing block and the options of rendering the triple in the attribute notation are examined. If the attribute-form presentation of the triple fails, the block is processed as a separate one, just as larger OPTIONAL blocks.

[4] A FILTER EXISTS block can be incorporated into a textual filter within a node, if it introduces a variable, followed by a filter on its value.

The visual query is a graph. Its construction starts with its ***node and edge creation*** for a ***single block*** identified in the query structure, followed by ***connecting the block visualizations***.

3.1 Query Block Visualization

We start the processing of a query block by considering all query variables, URIs and blank nodes that appear in the subject or object positions in the basic graph pattern (BGP) triples in the block as the ***node candidates*** in its graph presentation. The ***edges*** connecting the created node candidates are created based on the block BGP triples and are decorated by the triple predicates (the decorations can be property paths or variables).

To obtain a compact query presentation, we transform the obtained node candidate and edge structure, as follows:

1. the presentation of any triple with *rdf:type* property and a variable or URI in the object position is transformed into the ***class specification*** for the node of the triple subject (variable, URI or blank node); note that we do not apply the transformation for the class specifications that are blank nodes; we are also not able to account for more than one class specification for a subject node (in such a case the other class assertions stay in the visual shape),
2. variables (not URIs and not blank nodes) that are mentioned in a single object position within the block triples (and are not mentioned in any subject or predicate position within the block triples) are rendered into the ***attribute notation*** for the node corresponding to the subject of the respective triple [5] [6]. This also includes the case of single-triple OPTIONAL blocks that are rendered as optional attributes.

To fit the created query block node and edge structure into the visual query shape with a spanning tree consisting of the principal edges, we mark all created edges as ***required join links***, except when the edges create a non-tree structure; in such case some edges are marked as ***auxiliary***. If the graph corresponding to a block is not connected, add empty-labelled (++ in the concrete syntax) edges to create the connections.

Note. The decisions on the main node in the visual query corresponding to the query block and on exact ways of marking some of edges auxiliary or adding the empty-labelled edges can be taken arbitrarily as this does not change the query fragment semantics (some considerations regarding the choice of the main node of a query fragment shall arise from connecting the fragments at a later point).

Figure 2 illustrates a single-fragment SPARQL query (with a single-triple OPTIONAL sub-fragment) and its visualization by the outlined algorithm (handling the query selection list shall be addressed shortly).

[5] The visualizer can be instructed not to create attribute notation for triples corresponding to the object properties defined in the data schema.

[6] The attributes corresponding to the BIND-expressions located in a block shall be created at a later stage.

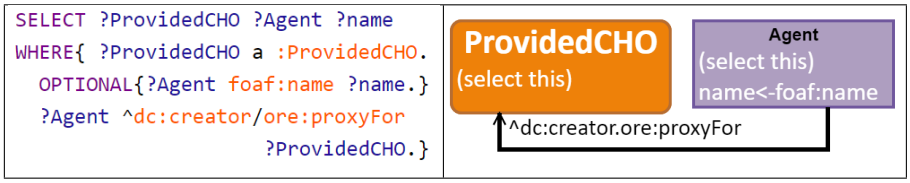


Fig. 2: Agents and their created CHOs that are provided to Europeana

3.2 Attributes, Filters and Aggregations

To complete the visual presentation of query blocks (as visual query fragments) the ***attribute lists*** for nodes are to be finalized, the ***filters*** are to be introduced, and the ***aggregates*** projected out of the main query and all its structural blocks that are subqueries are to be defined. The defined attribute and aggregate lists shall also describe the ***selection lists*** for the query and its subqueries. We arrange the process by traversing the query block graph (as observed in the textual SPARQL query) in a bottom-up manner since the attributes of an outer block may refer to the selection out of an inner block.

The ***attribute list*** for a node in a visual query fragment corresponding to a block shall involve:

1. the variables already presented in an attribute notation,
2. the variables computed as BIND-expressions within the fragment (each variable goes to a single node where it belongs most naturally), and
3. the variables that are projected out of the subquery-style fragments that are direct children of the current fragment; only those variables that are to be selected out of the current fragment are included.

The ***aggregates*** of nodes in a visual query fragment (that is the main query or is a subquery-style fragment) are created based on the aggregate expression binding in the selection list of the corresponding query block (the variables used as arguments to the binding should have been visualized in the created node attribute lists at this point).

The ***filters*** in the query nodes are created in accordance with the textual filter definition based on the computed attributes (some existential filters that involve a richer structure than a triple together with its value checking filter correspond to separate query fragments).

We perform an optimization to the created attribute structure to exclude from the node attribute lists those attributes that are used just in a single place within a BIND-expression, or an aggregate expression, or a filter, and that are not selected out of the query fragment themselves; these variables become ***implicit*** within the respective BIND-expression attribute, the aggregate expression, or the filter.

To ensure that the created attribute list defines the appropriate selection list for the query fragment (the attributes are selected by default), we mark all attributes that are not included in the selection list as helpers ({h} notation in the concrete syntax) and add explicit *(select this)* attributes for nodes whose variables are to be included in the select list (a node variable is not included in the select list by default).

3.3 Connecting Block Visualizations

The connection of (the visual fragments corresponding to) including blocks is based on identifying usage of the same variables in both the outer (the including) and the inner (the included) block. The primary connection between the blocks is based on just ***one variable corresponding to a node in each of the two blocks***. If there is no such variable (and there is no option to obtain one by reversing the earlier decisions to represent some variables as attributes), we introduce an empty-labelled edge (++) to connect some nodes within the blocks and do not consider further optimizations.

If such same-variable nodes, say N in the inner block's fragment and, say E, in the outer block's fragment exist, we initially draw an edge, labelled by '==' (the same-variable edge) between E and N (in this way we achieve the possibility to connect the fragments in any situation).

The principal idea of the block merging is to ***merge*** the nodes N and E into a ***single node in the visual query***, whenever this is possible. For the merging N should not have any attribute fields (except *(select this)*) and the inner block should remain non-empty after the removal of N. In this case we choose a node N' in the inner fragment (preferably, N' should be connected to N, otherwise an empty link with the ++ label is to be created) to become the ***principal node*** of the inner fragment. We consider the link connecting N (merged with E) and N' to be the connection between the outer and inner fragments and mark it with the type of the ***outer-to-inner block connection type*** (e.g., subquery/optional/not/union branch); mark other edges (if any) going from N to the other inner block nodes as auxiliary.

Figure 3 shows an example of merging nodes corresponding to the variable *?Agent*: the query fragments (a) and (b) together give the compound query (c).

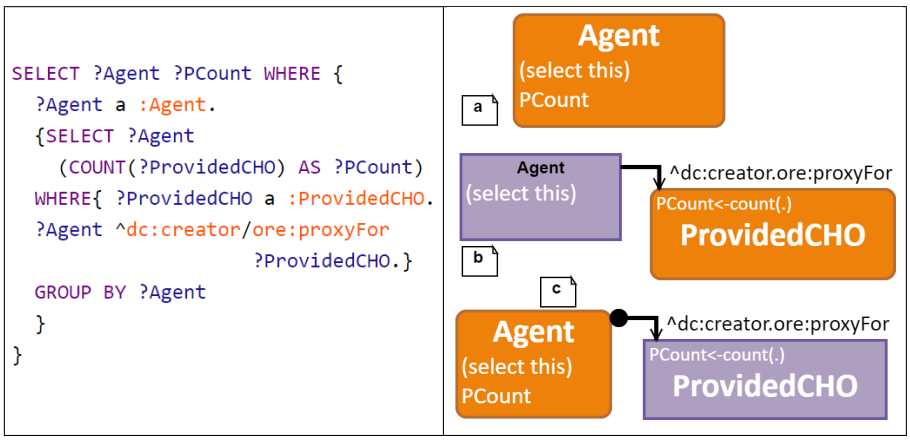


Fig. 3: Merging of nodes of including blocks

To optionally visualize other inter-block connections, if there are blocks X in Y in the query block structure, where X is (either directly or indirectly) above Y and there

are nodes E in X and N in Y, both marked with the same variable, connect E and N with an auxiliary edge, labelled by ==. Further on, if N has only one edge G in Y and N does not have any attribute (except *(select this)*), remove N and re-connect its edge G to E (as an auxiliary edge) instead of N.

After the block fragment merging, add the solution modifiers, as appropriate. In the case if a subquery has a slicing modifier (LIMIT or OFFSET), mark it a global subquery.

3.4 Implementation Notes

The SPARQL query visualization algorithm prototype has been implemented in Javascript programming language within the open source ViziQuer environment providing access possibility to the implementation[7]. The concrete syntax rendering of the query is provided by the API functions for visualizing the ViziQuer abstract syntax.

To ease the local ViziQuer server usage, a pre-built Docker environment with the tool is offered.

The visualization of a SPARQL query can be done within the visual query environment by copying the SPARQL query text into the SPARQL query pane and activating a 'Visualize SPARQL' context menu item.

The visualization of a SPARQL query is available in the context of any SPARQL endpoint for which the schema has been collected and the visual query environment has been created. There is an open-source tool available for retrieving the schema from a SPARQL endpoint[8] for the usage in the visual query tool.

4 Evaluation and Discussion

A simple notion of a successful visualization of a SPARQL query over a given data set would be to require that the visual query, when translated back into SPARQL and executed, would produce the same result as the execution of the original query. This notion may admit the dependency of the visualization and the visual query execution on the particular data set (e.g., if a property is known to have a maximum cardinality 1, both the visualization and visual query execution processes can possibly be simplified) and it can be contrasted with the mathematical notion of visualization success that would lead to a mathematically equivalent SPARQL query after the visualization and SPARQL query generation round-trip disregarding any data set specificity.

In the present work we consider the simple notion of the visualization success and work to expand the set of practical SPARQL queries that can be successfully visualized.

The current implementation of the visualization algorithm can be shown to successfully visualize the queries from the following query sets:

- the SPARQL queries generated from the visual queries in existing ViziQuer methodology and tool demonstrations, including [2, 3, 5], as well as the ViziQuer user manual (the "reverse" visualization of originally visual queries);

[7] https://github.com/LUMII-Syslab/viziquer

[8] https://github.com/LUMII-Syslab/obis-schemaextractor

- the example queries targeting the Europeana SPARQL endpoint [9] and described in its documentation [1];
- the example queries targeting the BNB SPARQL endpoint [10].

The live environments with Europeana and BNB example query visualizations are available from the paper's support page http://viziquer.lumii.lv/examples/sparql2viziquer. The chosen approach to consider the reverse visualization of the queries generated initially from the visual form is a natural one, as this would show the reversibility of the visual query implementation and show that the concepts of the visual query creation can be identified within SPARQL queries presented initially in the textual form.

Visualization of example query sets over different data endpoints indicate the possibility to obtain the visual form from a query that is expected to be understood by an end-user. We consider the addition of the visual form to an existing SPARQL query to bring another possible benefit in regarding the SPARQL query understandability (by comparing the comprehension of the textual query form alone to the comprehension of the textual query form together with the visual one). Still, the contribution of the visual notation to the SPARQL query comprehensibility remains to be evaluated.

Since the translation of a SPARQL query into the visual notation, as described in Section 3, goes on a construct-by-construct basis with very limited new construct introduction options, involving the optimizations related with the class and attribute form introduction in the visual presentation, the visual query would be generally of comparable or lower structural complexity than the textual query. Furthermore, the visual query presentation has appearance benefits if compared to the textual query presentation form by virtue of structuring the query across several interlinked graph nodes.

5 Related Work

A prominent UML-style visual SPARQL query creation environment is Optique VQs [12]. If compared to ViziQuer, the visual query constructs in Optique VQs are more limited, as they do not support subqueries, or the optional and negation modalities of join queries. There is also a much more limited expression language and expressions are not shown explicitly in the Optique VQs visual query presentation. Although the approach chosen by Optique is clearly relevant for usage of non-IT experts, the visual query presentation alone could not be regarded as a re-formulation of the textual SPARQL query. There are studies regarding textual SPARQL query rendering into a visual form in Optique VQs [11], however, our approach provides means for visualizing a much wider scope of the SPARQL queries.

The option of creating a visual presentation for a SPARQL query has been considered also in Sparqling [6], however, it is not using the UML-style presentation of the visual queries, and the scope of supported constructs is limited.

The ability of rendering a textual query visually is common also for the major relational database frameworks, however it is usually limited to simple join-filter-select queries that can be visually presented, as well as the visual presentation is not intended

[9] http://sparql.europeana.eu

[10] http://ldodds.github.io/bnb-queries/, the endpoint: https://bnb.data.bl.uk/sparql

to cover every aspect of the query definition (typically excluding any visual description for advanced expression computation, e.g., to describe the filters involved in the query).

The other major frameworks for visual SPARQL query creation, as [8, 9] do not provide explicit focus on obtaining a visual presentation from a textual SPARQL query expression.

6 Conclusions

The translation of a textual SPARQL query into a visual notation provides a new query viewing perspective, that can be beneficial in the query understanding. By contributing to the understandability of SPARQL queries, the query visualization has a potential to enhance the options for query sharing among the users and query reuse. The visualization of SPARQL queries would clearly enhance the options of learning and adopting the visual notation; it might be of help also in understanding and learning SPARQL itself.

The visual query environment allows further tuning of a query after its visualization, e.g., by adding the class information to some of the variables used in the query, in this way potentially further enhancing the end-user comprehension of a query.

An interesting perspective of using a SPARQL query visualization mechanism would be to integrate it with SPARQL query development tools, as SPARKLIS [7] for natural language-based query creation, or some form-based SPARQL query creation tool, as PepeSearch [13] or WYSIWYQ [10], to obtain a multi-modal SPARQL query management environment with integrated SPARQL query visualization option.

An important future work would be to assess the real benefits that the SPARQL query visualization could bring to an end-user; the potential end-users would be expected to be involved in such a study. As the provided SPARQL query visualization algorithm is primarily experience-based, a further study of its formal properties would be beneficial, as well.

In the general context of visual support to SPARQL query presentation, an interesting development avenue would be to also create a visual notation for SPARQL CONSTRUCT queries, coupled with their corresponding visualization tools.

7 Acknowledgements

This work has been partially supported by a Latvian Science Council Grant lzp-2020/2-0188 “Visual Ontology-Based Queries”.

References

1. Europeana sparql api, https://pro.europeana.eu/page/sparql, accessed on 24.09.2021
2. Čerāns, K., Bārzdiņš, J., Šostaks, A., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Sproģis, A.: Extended uml class diagram constructs for visual sparql queries in viziquer/web. In: VOILA@ISWC. pp. 87–98. No. 1947 in CEUR Workshop Proceedings (2017), http://ceur-ws.org/Vol-1947/paper08.pdf

3. Čerāns, K., Ovčiņņikova, J., Lāce, L., Hodakovska, J., Romāne, A., Grasmanis, M., Kalniņa, E., Sproģis, A., Šostaks, A.: Visual query environment over rdf data. In: Posters and Demos at SEMANTiCS 2019. pp. 156–160. No. 2451 in CEUR Workshop Proceedings (2019), http://ceur-ws.org/Vol-2451/paper-32.pdf
4. Čerāns, K., Šostaks, A., Bojārs, U., Bārzdiņš, J., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Sproģis, A.: Viziquer: a visual notation for rdf data analysis queries. In: Research Conference on Metadata and Semantics Research. pp. 50–62. No. 846 in CCIS, Springer (2018)
5. Čerāns, K., Šostaks, A., Bojārs, U., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Romāne, A., Sproģis, A., Bārzdiņš, J.: Viziquer: a web-based tool for visual diagrammatic queries over rdf data. In: The Semantic Web: ESWC 2018 Satellite Events. pp. 158–163. No. 11155 in LNCS, Springer (2018)
6. Di Bartolomeo, S., Pepe, G., Savo, D.F., Santarelli, V.: Sparqling: Painlessly drawing sparql queries over graphol ontologies. In: VOILA@ISWC. pp. 64–69. No. 2187 in CEUR Workshop Proceedings (2018), http://ceur-ws.org/Vol-2187/paper6.pdf
7. Ferré, S.: Sparklis: An expressive query builder for sparql endpoints with guidance in natural language. Semantic Web **8**(3), 405–418 (2017)
8. Haag, F., Lohmann, S., Siek, S., Ertl, T.: Queryvowl: Visual composition of sparql queries. In: The Semantic Web: ESWC 2015 Satellite Events. pp. 62–66. No. 9341 in LNCS, Springer (2015)
9. Kapourani, B., Fotopoulou, E., Papaspyros, D., Zafeiropoulos, A., Mouzakitis, S., Koussouris, S.: Propelling smes business intelligence through linked data production and consumption. In: On the Move to Meaningful Internet Systems: OTM 2015 Workshops. pp. 107–116. No. 9416 in LNCS, Springer (2015)
10. Khalili, A., Merono-Penuela, A.: Wysiwyq-what you see is what you query. In: VOILA@ISWC. pp. 123–130. No. 1947 in CEUR Workshop Proceedings (2017), http://ceur-ws.org/Vol-1947/paper11.pdf
11. Mumtaz, Y.: Collaborative Query Formulation in a Visual Query System. Master's thesis (2015)
12. Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., Horrocks, I.: Experiencing optiquevqs: a multi-paradigm and ontology-based visual query system for end users. Universal Access in the Information Society **15**(1), 129–152 (2016)
13. Vega-Gorgojo, G., Giese, M., Heggestøyl, S., Soylu, A., Waaler, A.: Pepesearch: semantic data for the masses. PloS one **11**(3), e0151573 (2016)
14. Zviedris, M., Barzdins, G.: Viziquer: a tool to explore and query sparql endpoints. In: The Semantic Web: Research and Applications. pp. 441–445. No. 6644 in LNCS, Springer (2011)

Fast Approximate Autocompletion for SPARQL Query Builders

Gabriel de la Parra and Aidan Hogan

DCC, Universidad de Chile & IMFD

Abstract. A number of interfaces have been proposed in recent years to help users build SPARQL queries, including textual editors with syntax highlighting and error correction, and visual editors that allow for drawing graph patterns using node and edge components. A common feature supported by such systems is autocompletion, which offers users suggestions for terms to insert into a query, potentially restricted by a keyword prefix. However, current systems either return irrelevant terms that will generate empty results, or return relevant terms but may time out while generating suggestions for complex queries. We propose an autocompletion technique based on a graph summary that aims to strike a balance by over-approximating relevant results in an efficient manner.

1 Introduction

The recent popularisation of knowledge graphs has introduced new users to the concept of representing and querying data through a graph abstraction [16]. Although a number of languages are now available for querying graphs – such as Cypher [11], Gremlin [23], SPARQL [15], etc. – users may not be familiar with such languages. Knowledge graphs are often used to represent diverse data that do not necessarily follow a schema [16], which makes querying them more difficult, even for users who are expert on the supported query language.

Prominent open knowledge graphs like Wikidata [27] already receive in the order of millions of queries per day from users over the Web [18]. A range of tools and techniques have thus been proposed to assist users to formulate queries over knowledge graphs more easily [24,1,17,9,7,19,2,2,13,22,3,8,18,25,26]. Among such systems, *autocompletion* is a key feature to improve usability. This feature allows users to select a term (often a property or a class) from a list of options, possibly matching a prefix that they have typed. Ideally, autocompletion should be *efficient*, enabling interactive query-building; and should return *relevant* results, avoiding terms that make no sense in the context of the current partial query.

We provide a real-world example of autocompletion in Figure 1 taken from the query editor provided by the Wikidata Query Service [18]. The user has created a variable `?x` and defined it to be an instance of (`wdt:P31`) film (`wd:Q11424`).

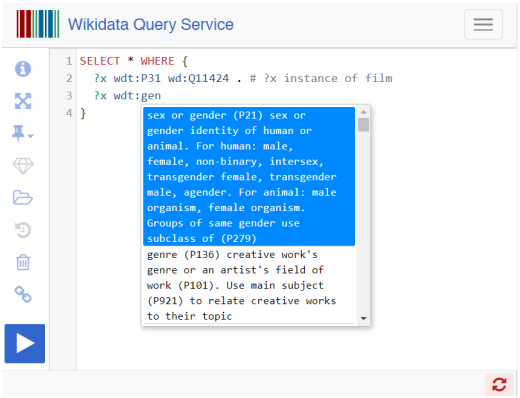


Fig. 1. Autocompletion for movie properties on the Wikidata Query Service's query editor (`https://query.wikidata.org`) based on the "`gen`" prefix

Next they wish to query for the genres of the film, and use the autocompletion feature typing "`gen`". However, the first suggestion is `sex or gender` (`wdt:P21`), which does not make much sense in the context of the current query. The second suggestion, `genre` (`wdt:P136`), is the one being sought.[1]

To avoid generating irrelevant suggestions, some query builder systems, such as Gosparqled [6] and RDF Explorer [26], only suggest terms that will generate non-empty results. In the context of the previous example, this is done by evaluating the following intermediate SPARQL query against the endpoint:

```
SELECT DISTINCT ?p WHERE { ?x wdt:P31 wd:Q11424 . ?x ?p ?o . }
```

in order to generate terms to replace `?p` that will ensure non-empty results. However, this query is quite expensive to evaluate, as it will typically involve checking all instances of film in the knowledge graph, and extracting the unique set of properties used to describe them. Theoretically speaking, the problem of knowing whether or not a particular term is an answer to such a query (a query with projection and a single Basic Graph Pattern (BGP)) over a knowledge-graph is NP-complete [20]. In practice, such intermediate queries may lead to timeouts, meaning that no suggestions are generated. Even in cases where results are successfully returned, these intermediate queries generate load on the server.

There thus exists a fundamental trade-off between the *efficiency* and *relevance* goals for SPARQL autocompletion. In this context, we find two types of techniques for autocompletion used in practice: those that return suggestions independently of the query context, which are efficient, but may generate irrelevant results (as used by the Wikidata Query Service), and those that only return suggestions leading to non-empty results, which are costly, but only generate relevant results (as used by RDF Explorer). A number of works have further explored techniques that try to strike a better balance between efficiency and rel-

[1] Prefixes used herein can be found at `http://prefix.cc`.

evance [7,6,12,4]. However, these approaches address autocompletion by trying to solve a problem that remains NP-complete in the general case.

For autocompletion, we believe that *efficiency* should take priority, and that *relevance* is a secondary-goal *so long as all relevant suggestions are returned*. In other words, we believe that it is key for the autocompletion technique to return results in a predictable (ideally sub-second) amount of time, no matter what the current query, and that it should return all relevant options, but it may return some irrelevant results as a trade-off (ideally as few as possible). We thus propose a *tractable* technique that filters suggestions according to the current query context, but that may *over-approximate* the set of relevant suggestions.

This paper then discusses our ongoing work on Fast Approximate Autocompletion for SPARQL (FAAS), along with some preliminary experiments on Wikidata. Section 2 describes related works on the topics of query builders and autocompletion for RDF/SPARQL. Section 3 introduces preliminaries and notation for RDF/SPARQL. Section 4 describes the novel technique we propose for autocompletion. Section 5 presents some experiments to evaluate our technique with respect to efficiency and relevance over Wikidata. Section 6 concludes the paper with a discussion of strengths, limitations, and future work.

2 Related Work

A range of tools, techniques and interfaces have been proposed in recent years to help users to interact with RDF datasets, including search engines, faceted browsing systems, graph profilers, visualisation tools, question answering services, query-by example systems, query editors, query builders, and more besides [10,5]. The most expressive tools are those that allow for constructing SPARQL queries, namely query editors and query builders. Query editors offer text- and form-based interfaces with aids for writing SPARQL queries (see, e.g., Figure 1); such systems include include Konduit [1], SPARQL Assist [19], Assisted SPARQL Editor [7], QUaTRO2 [2], Gosparqled [6], YASGUI [22], and the Wikidata Query Service [18]. Query builders allow for constructing a query through a visual abstraction, e.g., drawing a graph pattern; such systems include NIGHTLIGHT [24], RDF-GL [17], Smeagol [9], QueryVOWL [14], OptiqueVQS [25], SPARQLing [3], ViziQuer [8], and RDF Explorer [26].

Many such tools rely on generating suggestions through autocompletion mechanisms to assist the user in selecting terms of interest. Such a feature is particularly useful in the context of language-agnostic knowledge graphs, such as Wikidata, where numeric identifiers are used for entities and properties (e.g., using `wd:Q11424` for `film`). Of these systems, Konduit [1], Smeagol [9], Assisted SPARQL Editor [7], SPARQL Assist [19], QUaTRO2 [2], QueryVOWL [14], YASGUI [22], OptiqueVQS [25], Wikidata Query Service [18], and RDF Explorer [26] support some form of autocompletion. Of these, Smeagol [9], QUaTRO2 [2], Gosparqled [6] and RDF Explorer [26] avoid empty results. Assisted SPARQL Editor [7] provides approximated suggestions.

Various works have addressed autocompletion in the context of RDF/SPARQL. Campinas et al. [7] use a graph summary consisting of three layers: a dataset layer, a node collection layer, and an entity layer. Each layer is based on a quotient graph, at the level of datasets, classes and entities. Autocompletion is then fulfilled by translating the SPARQL query into a higher-level query over the graph summary. Gombos and Kiss [12] present an autocompletion approach that takes into consideration two properties: `rdf:type` to extract the type of entity and the properties compatible with those types, and `rdfs:range` to extract the possible types of the object variables. Rafes et al. [21] propose an autocompletion technique for SPARQL based on patterns seen previously in other users' queries. Bast et al. [4] recently proposed an autocompletion method based on query templates for generating ranked suggestions of entities that may include the context of the current query (without the triple being constructed). Given that these queries can be expensive to compute, the authors propose optimisations based on characteristic sets and caching techniques.

The most similar approaches to that which we propose are the works by Campinas et al. [7], Gombos and Kiss [12], and Bast et al. [4]. The main difference between our approach and that of Campinas et al. [7] and Bast et al. [4] is that their approaches consider evaluating the graph pattern on either a summarised version of the graph, or using particular optimisations; as we later discuss, both techniques are thus based on the NP-complete problem of graph homomorphism, and thus cannot guarantee efficiency in certain cases.[2] Our approach is based on a tractable (over-approximated version of the original) problem, which offers some theoretical guarantees of efficiency. Compared with Gombos and Kiss [12], there are some similarities with our technique, but both approaches are fundamentally different; for example, their approach relies on explicit `rdfs:range` definitions to be provided, and is "unidirectional", not considering domains for outgoing properties. Our approach does not rely on such definitions being provided.

3 Preliminaries

Before presenting our technique, we first present some brief, necessary preliminaries for RDF graphs and for SPARQL queries.

An RDF graph is based on three pairwise disjoint sets of RDF terms: IRIs ($\mathbf{I}$), literals ($\mathbf{L}$) and blank nodes ($\mathbf{B}$). An RDF triple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is a three-tuple of RDF terms, where s is called subject, p predicate and o object. An RDF graph G is then a finite set of RDF triples.

At the core of SPARQL queries lies the notion of a triple pattern, defined as $(s, p, o) \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$, where $\mathbf{V}$ denotes a set of variables disjoint from the RDF terms. Since blank nodes act as variables and subject literals cannot match any RDF triples, we will assume a simpler definition: $(s, p, o) \in (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$. A basic graph pattern B is then a set of triple patterns; equivalently we can think of a basic graph pattern

[2] In fact, the approach of Bast et al. [4] provides precise suggestions, and thus solves the original problem, which is known to be intractable [20].

as an RDF graph that permits variables in any position. We denote the set of variables appearing in a basic graph pattern B as $\text{vars}(B)$.

The evaluation of a basic graph pattern over an RDF graph yields a set of solution mappings. A solution mapping μ is a partial mapping $\mathbf{V} \to (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ from variables to RDF terms. We denote by $\text{dom}(\mu)$ the domain of μ, which is the set of variables for which μ is defined. We denote by $\mu(B)$ the image of B under μ; i.e., the result of replacing every variable $v \in \text{dom}(\mu) \cap \text{vars}(B)$ with $\mu(v)$ in B. The evaluation of a basic graph pattern B on an RDF graph G is then defined as $B(G) = \{\mu \mid \mu(B) \subseteq G \text{ and } \text{dom}(\mu) = \text{vars}(B)\}$; in other words, the evaluation returns all of the solution mappings that map the variables of B to the RDF terms of G such that the result is a sub-graph of G.

We are interested in generating suggestions for one variable at a time (what Campinas calls the "point of focus" [6]). Thus we introduce (single-variable) projection. Given a solution mapping μ and a variable $v \in \text{dom}(\mu)$, let μ_v denote the solution mapping such that $\text{dom}(\mu_v) = v$ and $\mu_v(v) = \mu(v)$, projecting (only) v from μ. Given a set of solution mappings M, we define $\pi_v(M) = \{\mu_v \mid \mu \in M\}$.

4 Fast Approximate Autocompletion for SPARQL

We now present our technique for fast approximate autocompletion in the context of SPARQL. We first show that autocompletion for SPARQL is a hard problem, which conflicts with the goal of efficiency in the general case, and motivates the need for approximate algorithms. We then introduce a relatively simple approach that over-approximates solutions for the problem, generating all relevant suggestions but possibly also some irrelevant suggestions.

4.1 Exact autocompletion is hard

The central problem of this paper is as follows:

PROBLEM:	Exact autocompletion
INPUT:	BGP B, variable $v \in \text{vars}(B)$, RDF graph G
OUTPUT:	$\pi_v(B(G))$

We begin by briefly showing that this problem cannot be solved efficiently. In particular, we show that the problem – a restricted case of the result presented by Pérez et al. [20] where one variable is projected – remains NP-hard.

Proposition 1. *Exact autocomplete is NP-hard.*

Proof. We present a polynomial-time reduction from the NP-complete problem of deciding graph homomorphism for directed graphs to the problem of exact autocompletion, which implies that the latter is NP-hard. Given the two directed graphs $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$, our goal is to decide whether or not there is a homomorphism from D_1 to D_2 (NP-complete). Define the basic graph pattern $B = \{(\nu(u), \mathsf{p}, \nu(v)) \mid (u, v) \in E_1\}$ where $\nu : V_1 \to \mathbf{V}$ is an injective

mapping from the vertices of V_1 to variables and $\mathsf{p} \in \mathbf{I}$ is an arbitrary IRI. Define the RDF graph $G = \{(\iota(u), \mathsf{p}, \iota(v)) \mid (u, v) \in E_2\}$ where $\iota : V_2 \to \mathbf{I}$ is an injective mapping from the vertices of V_2 to IRIs. Taking any variable $v \in \text{vars}(B)$, then there exists a homomorphism from D_1 to D_2 if and only if $\pi_v(B(G))$ is non-empty. This concludes the proof. □

In the interest of efficiency, we thus propose to compute approximate solutions for the autocompletion problem.

4.2 Over-approximating autocompletions

In our approach, we propose to ensure that all relevant suggestions be returned, but allow non-relevant suggestions be returned as well in order to trade relevance for efficiency. Specifically, we thus address the following problem:

PROBLEM:	Over-approximated autocompletion
INPUT:	BGP B, variable $v \in \text{vars}(B)$, RDF graph G
OUTPUT:	M such that $\pi_v(B(G)) \subseteq M$

Of course, there are some trivial solutions for M, such as mapping v to every RDF term in G. However, our goal will be to try to (efficiently) minimise the set of solutions in $M \setminus \pi_v(B(G))$, i.e., the false positives that lead to empty results. These are the solutions $\mu \in M$ such that $\mu(B)(G) = \emptyset$.

To further reduce and minimise the irrelevant results, we include some practical features, whereby:

1. A user can specify a prefix, such as "gen", that will be matched with keywords associated with suggestions (e.g., though a label or alias property).
2. Ranking is used to prioritise the suggestion of candidates.

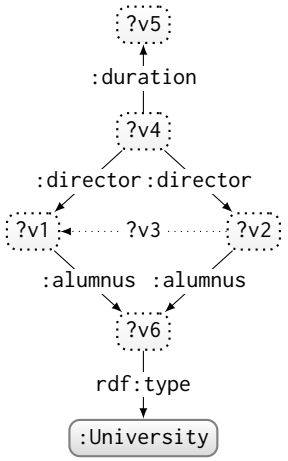
We will now describe the core of the technique we propose, which we call FAAS.

4.3 FAAS: core technique

From the RDF graph, we begin by extracting a summary to optimise suggestions generated by autocompletion. We assume here the definition of a type property IRI, which may be `rdf:type` or, in the case of Wikidata, `wdt:P31` (instance of). We will denote this IRI by the symbol ε for brevity.

Definition 1 (Autocompletion graph summary). *Let G be an RDF graph, p be an IRI, c and x be IRIs or blank nodes and y be an IRI, literal or blank node. We define the autocompletion graph summary as a tuple of six mappings:*

$$\begin{aligned}
\text{class}(x) &= \{c \mid (x, \varepsilon, c) \in G\} \\
\text{instance}(c) &= \{x \mid (x, \varepsilon, c) \in G\} \\
\text{incoming}(c) &= \{p \mid \exists x, y : \{(x, p, y), (y, \varepsilon, c)\} \subseteq G\} \\
\text{outgoing}(c) &= \{p \mid \exists x, y : \{(x, p, y), (x, \varepsilon, c)\} \subseteq G\} \\
\text{domain}(p) &= \{c \mid \exists x, y : \{(x, p, y), (x, \varepsilon, c)\} \subseteq G\} \\
\text{range}(p) &= \{c \mid \exists x, y : \{(x, p, y), (y, \varepsilon, c)\} \subseteq G\}
\end{aligned}$$



AUTOCOMPLETION GRAPH SUMMARY $\mathfrak{G}$ (sample)

domain(`:alumnus`)	= {`:Person`}
range(`:alumnus`)	= {`:School`, `:University`}
domain(`:director`)	= {`:Company`, `:Film`, `:Series`}
range(`:director`)	= {`:Person`}
domain(`:duration`)	= {`:Event`, `:Film`, `:Series`}
range(`:duration`)	= {`xsd:duration`}
domain(`:type`)	= {`rdfs:Resource`}
range(`:type`)	= {`rdfs:Class`}
...	= ...
incoming(`:Person`)	= {`:director`, `:parent`, `:sibling`}
outgoing(`:Person`)	= {`:alumnus`, `:parent`, `:sibling`}
...	= ...
instance(`:University`)	= {`:Aalborg`, ..., `:Zurich`}
...	= ...
class(`:University`)	= {`rdfs:Class`}
...	= ...

Fig. 2. Autocompletion example for BGP (left) and graph summary (right)

With respect to the previous definitions, we may note that $c \in \text{class}(x)$ if and only if $x \in \text{instance}(c)$, that $p \in \text{incoming}(c)$ if and only if $c \in \text{domain}(p)$, and that $p \in \text{outgoing}(c)$ if and only if $c \in \text{range}(p)$. Thus the summary contains some redundancy. Defining the mappings in this way, we then index on the argument, e.g., to quickly find instances by class, to find classes by instance, etc. If we wish to support autocompletion for datatype literals, we can consider adding a triple (l, ε, d) to denote that literal l is of type d. Furthermore, to ensure that all relevant results are returned, we assume that all nodes are declared to be an instance of some general class (e.g., `owl:Thing`, `rdfs:Resource`), or that all nodes can be returned if needed (akin to emulating a top-level class).

Given a basic graph pattern B, and a variable $v \in \text{vars}(B)$, the next step is to generate autocompleted suggestions for v using the summary of G. Let:

$$\text{s}(B) = \{x \mid \exists p, y : (x, p, y) \in B\}$$
$$\text{p}(B) = \{p \mid \exists x, y : (x, p, y) \in B\}$$
$$\text{o}(B) = \{y \mid \exists x, p : (x, p, y) \in B\}$$

denote the subject, predicate and object terms in B, respectively.

Our first goal is to generate candidate types for the nodes (subjects and objects) and then candidates for variable predicates in B. We begin with the example shown in Figure 2 before defining the algorithm. We will denote by B the basic graph pattern on the left, and by $\mathfrak{G}$ the autocompletion graph summary on the right. We begin by inferring the types of nodes, denoted with types[]. For constant nodes, we take the types from $\mathfrak{G}$.instance($\cdot$). For variables nodes, we intersect the types given in the basic graph pattern (if any), the domains of outgoing properties (if any), and the ranges of incoming properties (if any).

- types[?v1] = $\mathfrak{G}$.domain(:alumus) $\cap$ $\mathfrak{G}$.range(:director) = {:Person}
- types[?v2] = $\mathfrak{G}$.domain(:alumus) $\cap$ $\mathfrak{G}$.range(:director) = {:Person}
- types[?v4] = $\mathfrak{G}$.domain(:director)$\cap$$\mathfrak{G}$.domain(:duration) = {:Film, :Series}
- types[?v5] = $\mathfrak{G}$.range(:duration) = {xsd:duration}
- types[?v6] = {:University} $\cap$ $\mathfrak{G}$.range(:alumus) = {:University}
- types[:University] = $\mathfrak{G}$.class(:University) = {rdfs:Class}

Next we compute candidate properties for predicates in the basic graph pattern, denoted properties[]. In the case that a predicate is an IRI, its only candidate will be itself. Otherwise, it will be the intersection of all the incoming and outgoing properties for the subject and object classes; in case a subject/object has multiple classes as candidates, we take the union of their incoming/outgoing properties before intersecting as the subject/object may be *any* of the classes. This gives us the following candidates for predicates:

- properties[:alumnus] = {:alumnus}
- ...
- properties[?v3] = $\bigcup_{c \in \text{types[?v1]}} \mathfrak{G}.\text{incoming}(c) \cap \bigcup_{c \in \text{types[?v2]}} \mathfrak{G}.\text{outgoing}(c)$
  = {:director, :parent, :sibling} $\cap$ {:alumnus, :parent, :sibling}
  = {:parent, :sibling}.

We detail the process in Algorithm 1, which uses the following convention: we use $\{\star\}$ as syntax to represent the set of all classes/properties in a graph, and define the special intersection $\cap_\star$ such that $A \cap_\star \{\star\} = A$ and $\{\star\} \cap_\star \{\star\} = \{\star\}$. This avoids having to store all classes/properties in the initial set of candidates. The algorithm first establishes a set of candidate classes (denoted as the array types[]) for each node, and then a set of candidate values (denoted as the array properties[]) for each predicate, based on the information available in the basic graph pattern B and the graph summary $\mathfrak{G}$. Note the following:

1. If a node is a constant, or is given multiple specific types in the basic graph pattern, then it would suffice to select one type as the set of candidates is considered a disjunction. However, to avoid non-determinism and simplify the algorithm, we add all known classes to the candidate types for the node.
2. The algorithm is potentially recursive in that we could use the candidate predicates to refine the possible types for nodes, which could then refine the set of predicates, and so on. We opt for a single pass to ensure efficiency.

Once we have these candidates, we can then generate suggestions for autocompletion on a variable $v \in \text{vars}(B)$ based on the candidate types or properties returned by Algorithm 1. If the variable is in a predicate position, $v \in \text{p}(B)$, then we simply return properties[v] (if properties[v] = $\{\star\}$, then all predicates in G are suggested). Otherwise, we return the union of $\mathfrak{G}$.instance(c) for all $c \in$ types[x] (if types[v] = $\{\star\}$, then all nodes in G are suggested).[3]

[3] This implies that if a variable appears in a predicate and subject or object position, we will generate suggestions based on it appearing in the predicate position, as this will generate the most selective results.

We remark that the overall process is tractable: given an input RDF graph G with n triples, then $\mathfrak{G}$ can be computed in time $O(n)$ by creating six hashtables with $O(n)$ keys (assuming ideal hashing; note that the number of unique terms in G is at most $3n$, i.e., $O(n)$). Similarly, let m denote the number of triple patterns in B. Then we can use two hashtables with $O(m)$ keys to store the candidates for each term in B, where each term has at most $O(n)$ candidates. The candidates can themselves be stored as (nested) hashtables. Thus the intersections of candidate sets for each term takes $O(n)$ at each step, and there are at most three intersections (in the case of classes), giving us a complexity in the order of $O(mn)$ for type inference and autocompletion.[4] The key to tractability here is avoiding a recursive process when inferring classes and properties; otherwise we would be solving the basic graph pattern on a quotient graph, which is still NP-hard following the same reasoning as seen for Proposition 1.

We further remark that the technique returns all relevant results (and possibly more) due to the fact that the graph summary is computed from the data (rather than, e.g., relying on potentially incomplete or inaccurate `rdfs:domain` and `rdfs:range` definitions), that empty suggestions are returned only if the query yields no results, that classes for node variables are only ruled out if they have an incompatible incoming/outgoing property on an incident edge, and that candidate properties for predicate variables are only ruled out if they have an incident node whose possible classes are all incompatible with the domain/range of that property. However, we may additionally return irrelevant results.

4.4 Our technique: in practice

The procedure proposed in the previous section has some considerable practical weaknesses. For example, consider the query:

```
SELECT * WHERE { ?s ?p ?o . }
```

Autocompletions for `?s` or `?o` will return all nodes in the graph, while autocompletions for `?p` will return all predicates. In fact, such a query would not be uncommon in query editor and builder interfaces as a starting point for constructing a query. Hence we support two common heuristics used by systems that offer autocompletion: filtering by prefix/keyword, and ranking.

First, we allow users to type a prefix or keyword that will be matched against the text for a node (attached by label, alias or comment properties, as configured by the administrator); e.g., a user seeking autocompletions for `?s` may type "`aa`" and be suggested `:Aalborg`, `:Aardvark`, etc.; similarly if they seek autocompletions for `?p` with prefix "`d`", they will be suggested `:director`, `:duration`, etc. In cases where a fixed set of candidates can be found, prefixes can be combined with the previously discussed criteria to generate relevant suggestions.

Note that the user need not actually type a prefix, or may enter a prefix that still generates thousands of results. Thus, we also provide a ranking of

[4] In practice, one could expect better performance, taking time $O(n)$ to compute the autocompletion graph summary, and time $O(m+n)$ to compute candidates assuming that the number of classes and properties, in particular, is constant.

nodes and properties to prioritise results. For this, we currently apply PageRank on the directed graph induced by the RDF graph to rank node suggestions, and we count the number of triples in which predicates appear to rank property suggestions. Instead of generating all results, we rather paginate the suggestions.

4.5 Prototype implementation

We have developed a prototype implementation in C#. The implementation takes as input a set of type properties, and a set of textual properties used for prefix and keyword matching. It then uses custom scripts to extract the autocompletion graph summary (using external sorting rather than hashing, with $O(n \log n)$ performance but fewer random accesses on disk). The mappings are implemented as two inverted indexes in Lucene separated by nodes and properties (some elements may be indexed in both). The inverted indexes further include text indexing on the textual properties, which supports wildcards (e.g., "`aa*`" for prefix search), and ranking boosts based on PageRank and property frequency, respectively. Boosts are multiplied by TF-IDF-based relevance scores in case that a prefix or keyword is provided by the user.

4.6 Limitations

There are a number of limitations of our proposed technique and current prototype. First, we focus on autocompletion within a given basic graph pattern. Though we can support other query features such as `UNION`, `GROUP BY`, etc., by focusing on generating autocompletions for inner basic graph patterns, features such as property paths are not directly supportable in this manner. Second, our autocompletion framework depends on an index over a graph summary, meaning that it cannot be directly applied over an endpoint; while specialised indexes help improve performance, such an approach generates an additional cost in terms of downloading the dump and processing it locally, and introduces the question of keeping the graph summary up-to-date with respect to remote changes, which we currently do not consider. Third, our approach may generate irrelevant suggestions that lead to empty results, and may even be slower in *some* cases (in particular, involving selective queries) than computing the exact results for a variable over the remote endpoint; for this reason, we propose to evaluate autocompletions locally and over the endpoint in parallel, and in case the endpoint returns within a fixed amount of time, to use the exact results, otherwise deferring to the approximate result.

5 Experiments

We now present the results of some initial experiments for our autocompletion prototype. In particular, we present experiments for autocompletions over Wikidata. The baseline that we currently consider involves computing the exact

results for a variable on the public Wikidata Query Service (WDQS). We consider two main aspects: efficiency and relevance. For efficiency, we will measure response times in comparison to the baseline. For relevance, we will measure the precision of the suggestions generated as the ratio of results returned by our system that are also returned by the endpoint.

Our experiments are based on the May 23rd 2020 truthy dump of Wikidata, which we preprocessed to filter non-English labels, descriptions, etc., leaving a dataset of 1.098 billion triples. Experiments were run on a personal computer with an i7-4600M CPU @ 2.9 GHz and 16 GB of RAM, running on Windows 10. In terms of preprocessing, computing the ranking, extracting the autocompletion graph summary and indexing all of the data took 75 hours. The index size was 9.7 GB, describing 9.7 million nodes and 7,559 properties.

In terms of experiments, to the best of our knowledge there is no existing benchmark for autocompletion. Hence we created a benchmark considering four query templates, as shown in the following:

```
1 SELECT DISTINCT ?p WHERE { ?v1 :p ?v2 . ?v1 ?q ?v3 }
2 SELECT DISTINCT ?p WHERE { ?v1 :p ?v2 . ?v2 ?q ?v3 }
3 SELECT DISTINCT ?p WHERE { ?v1 :p ?v2 . ?v3 ?q ?v1 }
4 SELECT DISTINCT ?p WHERE { ?v1 :p ?v2 . ?v3 ?q ?v2 }
```

Here `:p` represents a constant property that we select based on a weighted sampling of 67 distinct properties in Wikidata, taking a mix of high and low values for the number of triples, number of domain classes, and the number of range classes associated with the property. These queries exhibit common types of joins, as are often encountered in intermediate queries under construction; they are also the types of queries that the endpoint struggles with in practice. Given that the WDQS endpoint frequently times out for node suggestions, where our local index rather paginates ranked results, to enable a fairer comparison we focus in these initial experiments on autocompleting `?q`.

We present the results in Figure 3, comparing the times for FAAS (our proposal) and WDQS, with the x-axis representing the percentile, and the y-axis representing the runtime, considering $67 \times 4 = 268$ queries. We see that FAAS is in general considerably faster than WDQS. The plateau seen for WDQS refers to queries that time out, where only 36% were processed successfully within the timeout. Given that it is difficult to see the times for FAAS, we present the results alone in Figure 4. We see that all autocompletions were processed within 3 seconds, while 35% of the queries were processed within 2 seconds, and 8% were processed within 1 second. While ideally the runtimes would be a bit lower (i.e., consistently sub-second), we consider that waiting up to 3 seconds is acceptable (particularly considering the alternative of generating no suggestions), and could be lowered with additional optimisations in future.

A key difference between WDQS and FAAS that enables the superior performance of the latter is that FAAS over-approximates suggestions while WDQS computes exact suggestions. For the queries that returned results from WDQS,

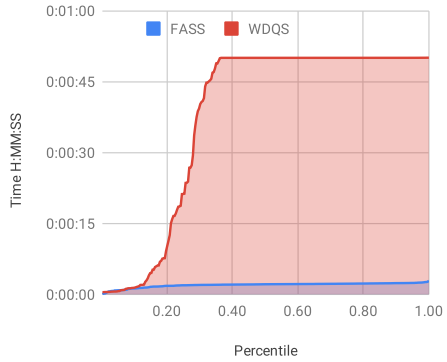


Fig. 3. Times: FAAS vs. WDQS

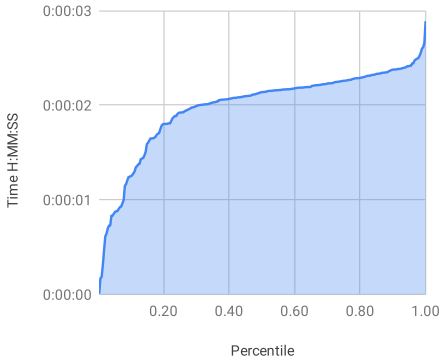


Fig. 4. Times: FAAS only

we thus measured the precision, where we found that across all such queries, the minimum precision was 0.04, the median precision was 0.21, and the maximum precision was 0.59. Thus in a typical case, we could expect roughly 1/5 of the suggestions returned by FAAS to return non-empty results when replacing the variable. We also measured the recall, which ranged from 0.92 to 1.00, with a median of 1.00; since we over-approximate results, we expect the recall to always be 1.00, where we confirmed that the cases with imperfect recall were caused by updates to the remote WDQS after the dump we use was published.

6 Conclusions

In this paper, we have described our ongoing work on a tractable technique for autocompletion of SPARQL query variables, which can be useful for query editors and query builders. Tractability is enabled by over-approximating the suggestions that lead to non-empty results. We have implemented a prototype based on Lucene, where initial experiments show promising results in terms of efficiency (all autocompletions are computed in less than 3 seconds, even without prefix or keyword search), at the cost of precision (0.21 in the median case, i.e., we return about 5 times more results than are actually relevant).

Key challenges for future work involve improving efficiency further (to get below 1 second, at least) while further increasing precision. Balancing the two appears tricky: for example, precision could be improved by adding a limited depth of recursion to Algorithm 1, but may lead to slower times. Another promising direction might be to look into ranking results by relevance to the query, which would reduce the impact of low precision on usability if the first results that appear will be relevant: though we do consider some ranking mechanisms, we do not consider the context of the query for ranking, which could be explored in future work. For instance, our algorithm will still return `sex or gender` (`wdt:P21`) as a suggestion for the motivating scenario posed in Figure 1: at the time of writing, four movies in Wikidata (presumably due to noise) have this property defined; in future, it could be ranked low based on being defined for few movies.

Another alternative would be to look at other variants of graph summary, for example, to include statistics that weight relations (e.g., to indicate how many instances of a class use a "domain" property, which might be helpful for query-specific ranking, as mentioned before), or to model direct relationships between properties that co-occur on nodes (e.g., to support additional features, such as property paths). Another challenge is keeping the graph summary up-to-date with respect to the knowledge graph, where computing the summary on a dump currently takes several days, and where we missed relevant suggestions due to changes in the live query service since the summary was computed; a possible solution would be to look into incremental updates of the summary.

We further plan to experiment with additional datasets, queries and baselines and have integrated our technique with RDF Explorer [26] for usability testing.

Please see `https://github.com/gabrieldelaparra/SPARQLforHumans` for code and additional material, including the queries used for experiments.

Acknowledgements This work was supported by ANID – Millennium Science Initiative Program – Code ICN17_002. Hogan was supported by Fondecyt Grant No. 1181896. We thank the reviewers for their helpful feedback; many of the interesting suggestions for future work were provided by them.

References

1. O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop. In *Visual Interfaces to the Social and Semantic Web (VISSW)*. ACM Press, 2010.
2. B. Balis, T. Grabiec, and M. Bubak. Domain-Driven Visual Query Formulation over RDF Data Sets. In *Parallel Processing and Applied Mathematics (PPAM)*, pages 293–301. Springer, 2013.
3. S. D. Bartolomeo, G. Pepe, D. F. Savo, and V. Santarelli. Sparqling: Painlessly Drawing SPARQL Queries over Graphol Ontologies. In *Visualization and Interaction for Ontologies and Linked Data (VOILA)*, pages 64–69, 2018.
4. H. Bast, J. Kalmbach, T. Klumpp, F. Kramer, and N. Schnelle. Efficient SPARQL autocompletion via SPARQL. *CoRR*, abs/2104.14595, 2021.
5. N. Bikakis and T. Sellis. Exploration and visualization in the web of big linked data: A survey of the state of the art. *arXiv preprint arXiv:1601.08059*, 2016.
6. S. Campinas. Live SPARQL Auto-Completion. In *ISWC 2014 Posters & Demos*, volume 1272 of *CEUR*, pages 477–480. CEUR-WS.org, 2014.
7. S. Campinas, T. Perry, D. Ceccarelli, R. Delbru, and G. Tummarello. Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In *International Workshop on Database and Expert Systems Applications (DEXA)*, pages 261–266. IEEE Computer Society, 2012.
8. K. Cerans, A. Sostaks, U. Bojars, J. Ovcinnikova, L. Lace, M. Grasmanis, A. Romane, A. Sprogis, and J. Barzdins. ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data. In *European Semantic Web Conference (ESWC)*, pages 158–163, 2018.
9. A. Clemmer and S. Davies. Smeagol: a "specific-to-general" semantic web query interface paradigm for novices. In *Database and Expert Systems Applications (DEXA)*, pages 288–302. Springer, 2011.

10. A.-S. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2):89–124, 2011.
11. N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An Evolving Query Language for Property Graphs. In *SIGMOD International Conference on Management of Data*, pages 1433–1445. ACM, 2018.
12. G. Gombos and A. Kiss. Federated Query Evaluation Supported by SPARQL Recommendation. In S. Yamamoto, editor, *Human Interface and the Management of Information: Information, Design and Interaction*, volume 9734 of *LNCS*, pages 263–274. Springer, 2016.
13. P. Grafkin, M. Mironov, M. Fellmann, B. Lantow, K. Sandkuhl, and A. V. Smirnov. Sparql query builders: Overview and comparison. In *BIR Workshops*, 2016.
14. F. Haag, S. Lohmann, S. Siek, and T. Ertl. QueryVOWL: A Visual Query Notation for Linked Data. In *Extended Semantic Web Conference (ESWC)*, pages 387–402. Springer, 2015.
15. S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, 2013. `https://www.w3.org/TR/sparql11-query/`.
16. A. Hogan et al. Knowledge Graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.
17. F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*, pages 87–116. Springer, 2010.
18. S. Malyshev, M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *International Semantic Web Conference (ISWC)*, pages 376–394. Springer, 2018.
19. E. L. McCarthy, B. P. Vandervalk, and M. Wilkinson. SPARQL Assist language-neutral query composer. *BMC Bioinformatics*, 13(S-1):S2, 2012.
20. J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.
21. K. Rafes, S. Abiteboul, S. C. Boulakia, and B. Rance. Designing Scientific SPARQL Queries Using Autocompletion by Snippets. In *IEEE International Conference on e-Science*, pages 234–244. IEEE, 2018.
22. L. Rietveld and R. Hoekstra. The YASGUI family of SPARQL clients. *Semantic Web*, 8(3):373–383, 2017.
23. M. A. Rodriguez. The Gremlin graph traversal machine and language (invited talk). In *Symposium on Database Programming Languages*, pages 1–10. ACM, 2015.
24. P. R. Smart, A. Russell, D. Braines, Y. Kalfoglou, J. Bao, and N. R. Shadbolt. A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. In *Knowledge Engineering and Knowledge Management (EKAW)*, pages 275–291. Springer, 2008.
25. A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie, and I. Horrocks. OptiqueVQS: A visual query system over ontologies for industry. *Semantic Web*, 9(5):627–660, 2018.
26. H. Vargas, C. B. Aranda, A. Hogan, and C. López. RDF Explorer: A Visual SPARQL Query Builder. In *International Semantic Web Conference (ISWC)*, pages 647–663. Springer, 2019.
27. D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Algorithm 1: Type and property inference

Input: Basic graph pattern B, graph summary $\mathfrak{G}$
Output: Candidate types for $\mathrm{s}(B) \cup \mathrm{o}(B)$, Candidate properties for $\mathrm{p}(B)$
initialise types[]; properties[];
for $x \in s(B) \cup o(B)$ **do**
  if $x \in \mathbf{I} \cup \mathbf{L}$ **then**
    types[x] $\leftarrow \mathfrak{G}$.class(x);
  else
    $C_x \leftarrow \{\star\}$;
    if $\exists c \in \mathbf{I} : (x, \varepsilon, c) \in B$ **then**
      $C_x \leftarrow \{c' \in \mathbf{I} \mid (x, \varepsilon, c') \in B\}$;
    end
    for $(x, p, o) \in B : p \in \mathbf{I}$ **do**
      $C_x \leftarrow C_x \cap_\star \mathfrak{G}$.domain($p$);
    end
    for $(s, p, x) \in B : p \in \mathbf{I}$ **do**
      $C_x \leftarrow C_x \cap_\star \mathfrak{G}$.range($p$);
    end
    types[x] $\leftarrow C_x$;
  end
end
for $p \in p(B)$ **do**
  if $p \in \mathbf{I}$ **then**
    properties[p] $\leftarrow \{p\}$;
  else
    $C_p \leftarrow \{\star\}$;
    for $(s, p, o) \in B : \text{types}[s] \neq \{\star\}$ **do**
      $C_p \leftarrow C_p \cap_\star \bigcup_{c \in \text{types}[s]} \mathfrak{G}$.outgoing($c$);
    end
    for $(s, p, o) \in B : \text{types}[o] \neq \{\star\}$ **do**
      $C_p \leftarrow C_p \cap_\star \bigcup_{c \in \text{types}[o]} \mathfrak{G}$.incoming($c$);
    end
    properties[p] $\leftarrow C_p$;
  end
end
return types[], properties[]

RepOSE-CTab - A Protégé Plugin for Completing Ontologies

Zlatan Dragisic[1], Ying Li[1], and Patrick Lambrix[1,2,3]

[1] Department of Computer and Information Science, Linköping University, Sweden
[2] The Swedish e-Science Research Centre, Linköping University, Sweden
[3] Department of Building Engineering, Energy Systems and Sustainability Science, University of Gävle, Sweden
patrick.lambrix@liu.se

Abstract. As the quality of ontologies plays an important role in supporting semantically-enabled applications, defining concepts as well as their relations correctly and completely is crucial when developing an ontology. In this paper we introduce a Protégé plugin for extending ontologies, which guides Protégé users through the addition of new concepts as well as their instances and axioms in which they participate in a semi-automatic way. Furthermore, the tool suggests additional subsumption axioms that a user can validate and, if appropriate, add to the ontology to make the ontology more complete.[1]

1 Introduction

Developing ontologies is not an easy task and when ontology developers add new concepts to an ontology they may not be as complete as they could be. When ontologies are not complete, it has an effect on semantically-enabled applications using these ontologies as valid conclusions may be missed [2]. For instance, in [3] an example is given where a missing subsumption relation leads to the fact that ontology-based search in PubMed misses more than half of the query results.

To improve the completeness of ontologies during development, we present a plugin for Protégé [4] that supports a user when adding new concepts to an ontology. The tool requests information to be filled out in terms of concept names, instances and super-concepts and shows the consequences, e.g., in terms of the subsumption hierarchy. Compared to the traditional way of adding concepts and instances in ontology editors, this plugin introduces an extra step, called completion, based on earlier work on the RepOSE tool [3]. In this step the plugin suggests additional subsumption axioms that the user can validate and, if appropriate, add to the ontology to make the ontology more complete. Although it may seem that when knowledge engineers and domain experts add knowledge in the form of subsumption axioms, they add all knowledge they know, this may not always be the case. For instance, [3] describes an experiment with the ontologies

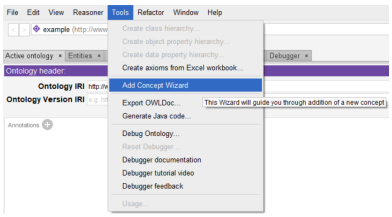


Fig. 1. The RepOSE-CTab as *Add Concept Wizard* in the tools menu.

of the Anatomy track of the Ontology Alignment Evaluation Initiative [1] where 94 subsumption axioms were initially added to the Adult Mouse Anatomy ontology, but using the RepOSE completion tool 47 additional subsumption axioms that could not be derived from the ontology and the 94 axioms, were found. For the NCI Human Anatomy ontology, 58 subsumption axioms were added initially and 10 additional axioms were found using completion.

In terms of the design space dimensions in [6] the **purpose** of the tool is to support a user when adding new concepts to an ontology. It focuses on $\mathcal{EL}$ ontologies, although for ontologies defined in more expressive knowledge representation languages, the tool can be used for the taxonomic part, i.e., subsumption relations between the named concepts representing ontological concepts. The main purpose of the tool is *creating and managing* as defined in [6] as it is essentially part of an ontology development tool. However, there is also a minor aspect of *learning and understanding* as a user needs to understand the concepts and their contexts to be able to make informed decisions about what axioms to add. The **users** of the tool are ontology developers. This usually requires expertise in ontology engineering as well as in the domain that is being modeled in the ontology.

In section 2 we describe our tool and in section 3 we show the use of the tool on a small example.

2 RepOSE-CTab

Our tool is a plugin for Protégé 5.0 (and tested for Protégé 5.0 and 5.1). It is Java based and uses the OWL API[2], the Protégé package for accessing and handling OWL ontologies and the Protégé editor. It is installed by copying a jar file (AddConcept) into the plugin directory of the existing main Protégé folder. Then upon a new start, Protégé will find the tool and create a new tab for it. The tool is then found in the tools menu as *Add Concept Wizard* (Figure 1).

The general process of adding a new concept as well as their subsumption relations and instances with the assistance of this Add Concept Wizard is as follows. In a first step, the user types the new concept's name and selects its super-concepts from the concepts list. When selecting the super-concepts, the

[2] http://owlapi.sourceforge.net/

tool provides information such as super-concepts, sub-concepts and disjoint concepts. This allows the user to get an overview of the possible context of the concept and may be helpful in deciding, e.g., to add disjointness or additional subsumption axioms. Upon providing the super-concepts of the new concept, the corresponding subsumption axioms are generated. Most of this functionality is common in ontology editors.

Furthermore, in a second step a unique feature not found in current ontology editors is implemented. The tool generates suggestions for additional subsumption axioms to add to the ontology. This step is called completing in [2] and is an abductive reasoning problem. The algorithm for generating these suggestions is explained, e.g., in [3]. For the implementation in this plugin the tool generates source and target sets for each axiom $\alpha \sqsubseteq \beta$. The source set contains the super-concepts of α and the target set contains the sub-concepts of β. We know then that adding any $\alpha_s \sqsubseteq \beta_t$ with α_s in the source set and β_t in the target set, to the ontology will make $\alpha \sqsubseteq \beta$ logically derivable from the ontology. However, the set of logical solutions may contain solutions that are not correct according to the domain. It is, therefore, important that a domain expert validates these $\alpha_s \sqsubseteq \beta_t$ to make sure that no wrong knowledge is added to the ontology.[3]

The source and target sets for each axiom are shown to the user who can decide on using such a new axiom by clicking on a concept in the source set and a concept in the target set and validate it or by validating it from a list. Further, when such new axiom is added, new knowledge is added to the ontology and there may be new completions possible. Therefore, this step is iterated.

In [2] a relation *more complete* is defined between different repairs, which, in this case, refers to different sets of axioms that are used to add knowledge to the ontology.[4] A repair R1 is more complete than another repair R2 if every correct statement according to the domain that can be derived from the ontology with the addition of the axioms in R2, can also be derived from the ontology with the addition of the axioms in R1, and there is correct statement according to the domain that can be derived from the ontology with the addition of the axioms in R1, but that cannot be derived from the ontology with the addition of the axioms in R2. Essentially, the new ontology based on R1 contains more correct knowledge than the new ontology based on R2 and thus is 'more complete'. Based on this definition, we can then say that, if an axiom is added in the second step of the process, then the new ontology after the second step is more complete than the ontology after the first step of the process and more complete than the ontology before we started adding a new concept.

[3] A similar issue appears in a dual problem of completing, i.e., debugging. While in completing logical solutions may add axioms that are not correct according to the domain, in debugging logical solutions may remove correct knowledge from the ontologies or mappings [5]. Therefore, in both cases a domain expert needs to validate the logical solutions.

[4] The definition in [2] takes also into account removal of axioms, but we use a simplified version only dealing with addition of axioms as the plugin only deals with adding concepts.

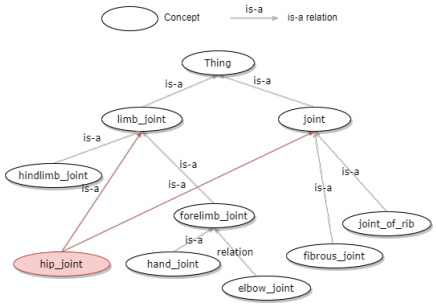


Fig. 2. A piece of the Adult Mouse Anatomy ontology.

Finally, the user can then also add existing instances for the new concept. As a help the user can obtain the instances of super-concepts or search for instances. Once all the relevant information is provided, all axioms and instances are listed in the final step. After that, the new concept as well as their instances and relevant axioms are added into the ontology.

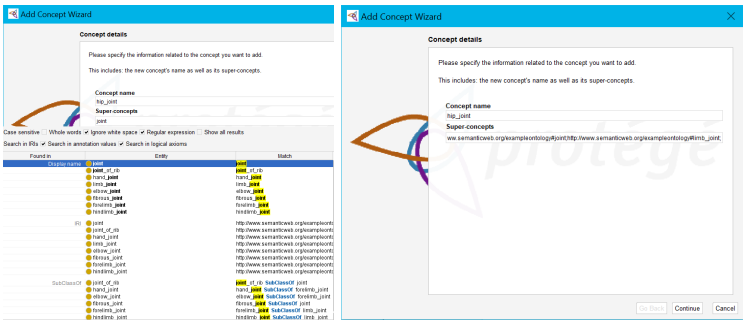
3 Example run

In the following we run through a scenario showing how the plugin guides a user when adding concepts and subsumption relations related to that concept to an ontology. We use an example inspired by the Adult Mouse Anatomy (`http://www.informatics.jax.org/vocab/gxd/ma_ontology/`, part of the Gene Expression Database) that is used in the Anatomy and Interactive tracks of the Ontology Alignment Evaluation Initiative [1].

Fig. 2 shows a small piece of the ontology and it is this piece that we will extend. We assume that the user wants to add the new concept *hip_joint* and the user knows that the existing concepts *joint* and *limb_joint* are super-concepts of the new concept. The user will use the plugin to add this knowledge to the ontology.

After loading the target ontology into Protégé and starting the built-in reasoner, the user can click on *Add Concept Wizard* in the tools menu to start the guide to concept and axiom addition. First, the user is asked to enter the concept name and select super-concepts from the list of concepts in the ontology. As an additional help, when typing the name of a super-concept, information about these concepts, such as disjoint concepts, super-concepts and sub-concepts, is shown (Figure 3a). In this example, the user uses *hip_joint* as the concept name, and *joint* and *limb_joint* as its super-concepts (Figure 3b). This means that the axioms $hip\_joint \sqsubseteq joint$ and $hip\_joint \sqsubseteq limb\_joint$ should be derivable from the extended ontology.

In the next step completion is performed. The tool proposes other axioms to add to the ontology that make $hip\_joint \sqsubseteq joint$ and $hip\_joint \sqsubseteq limb\_joint$ derivable, and would add more new knowledge to the ontology. As explained



(a) (b)

Fig. 3. Concept name and super-concepts.

earlier, it does so by creating source and target sets for these axioms and asks the user to validate axioms $\alpha_s \sqsubseteq \beta_t$ with α_s in the source set and β_t in the target set. In our example, for *hip_joint* $\sqsubseteq$ *limb_joint*, the source set is {*hip_joint*} and the target set is {*limb_joint*, *hindlimb_joint*, *forelimb_joint*, *hand_joint*, *elbow_joint*} (Fig. 4a). The user recognizes that *hip_joint* $\sqsubseteq$ *hindlimb_joint* and chooses and validates this relation. When adding the *hip_joint* $\sqsubseteq$ *hindlimb_joint* into the ontology, *hip_joint* $\sqsubseteq$ *limb_joint* can be derived from the ontology and thus is redundant. The user can remove this redundant axiom by clicking the validate relation button, uncheck the relation *hip_joint* $\sqsubseteq$ *limb_joint* in the validated relation list and then click the validate button to finish.

Similarly, the tool computes the source and target sets for *hip_joint* $\sqsubseteq$ *joint*. The source set is {*hip_joint*} and the target set is {*joint*, *fibrous_joint*, *joint_of_rib*} (Fig. 4b). The additional suggestions from the tool are not retained. After this first iteration we thus have the original axiom to add *hip_joint* $\sqsubseteq$ *joint* and the new *hip_joint* $\sqsubseteq$ *hindlimb_joint*. As we have added a new axiom we start a new iteration.

Fig. 4c shows the new source ({*hip_joint*, *limb_joint*, *hindlimb_joint*,*joint*}) and target ({*joint*, *fibrous_joint*, *joint_of_rib*, *hip_joint*}) sets of *hip_joint* $\sqsubseteq$ *joint*. The user may note that *limb_joint* $\sqsubseteq$ *joint* and choose and validate this relation. Then, the iteration is ended by the user. As before, the user can uncheck redundant relations in the validated relation list.

The plugin also helps adding instances to the ontology, but in this example we do not have any.

In the final step, all axioms that we validated are shown to the user again (Fig. 5a). Upon finishing, the new concept *hip_joint* and the remaining axioms *hip_joint* $\sqsubseteq$ *hindlimb_joint* and *limb_joint* $\sqsubseteq$ *joint* are added to the ontology (original ontology in Fig. 5b and new ontology in Fig. 5c).

The new ontology contains the new concept and the axioms based on the super-concepts are derivable. However, we also added additional knowledge during the completion step. For instance, *elbow_joint* $\sqsubseteq$ *joint* is now also derivable

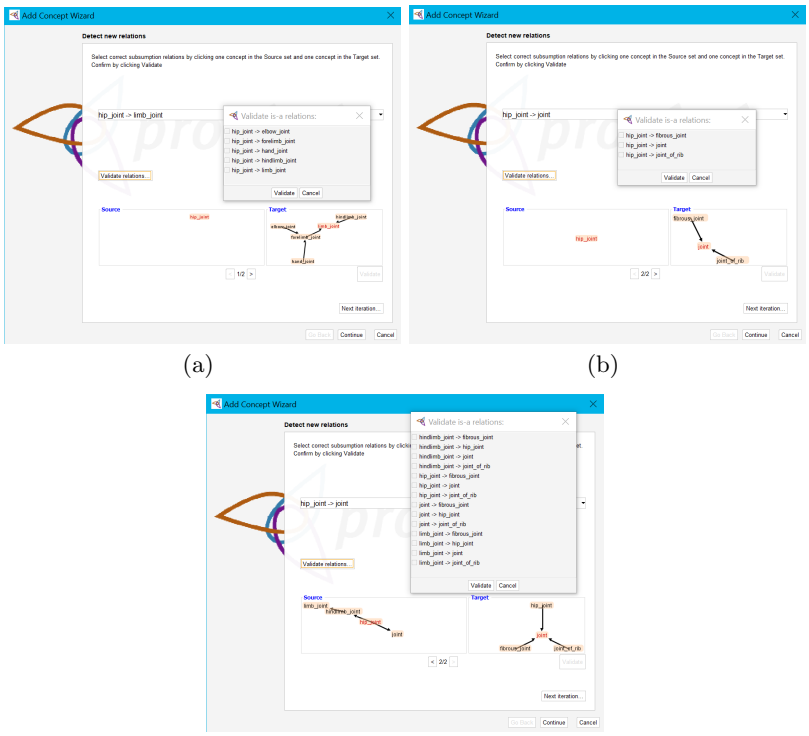


Fig. 4. Source and Target sets for different axioms and iterations.

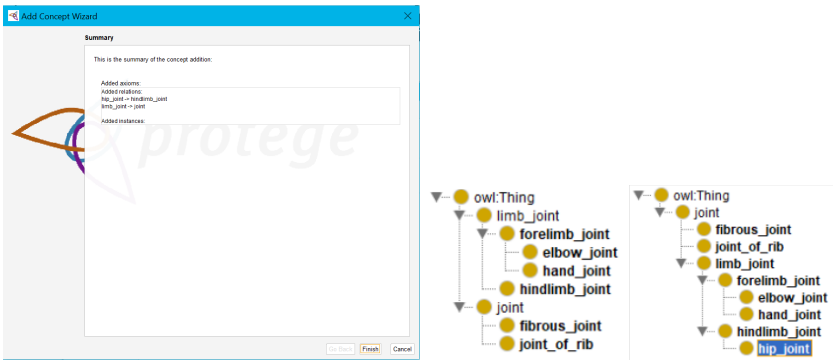
from the ontology. The extension of the ontology using the axioms in the completing step is therefore more complete than the extension with the axioms based on the original super-concepts assigned by the user in the first step.

4 Conclusion

In this paper, we introduced a plugin for the ontology editor Protégé, which guides the Protégé user through extending an ontology by adding new concepts as well as their relations and instances in a semi-automatic way. The use of the plugin leads to more complete ways to extend the ontology than just adding the concepts.

5 Availability

The plugin is available at:
`https://www.ida.liu.se/~patla00/research/RepOSE/downloads.html`.



(a) The validated subsumption axioms. (b) (c)

Fig. 5. Adding subsumption axioms.

Acknowledgements. This work has been financially supported by the Swedish e-Science Research Centre (SeRC), the Swedish National Graduate School in Computer Science (CUGS), the Swedish Research Council (Vetenskapsrådet, dnr 2018-04147), the Horizon 2020 project SPIRIT (grant agreement No 786993), and the Swedish Agency for Economic and Regional Growth (Tillväxtverket, 20201438).

References

1. Dragisic, Z., Ivanova, V., Li, H., Lambrix, P.: Experiences from the anatomy track in the ontology alignment evaluation initiative. Journal of Biomedical Semantics **8**(1), 56:1–56:28 (2017). https://doi.org/10.1186/s13326-017-0166-5
2. Lambrix, P.: Completing and debugging ontologies: state of the art and challenges (2020), arXiv:1908.03171
3. Lambrix, P., Wei-Kleiner, F., Dragisic, Z.: Completing the is-a structure in light-weight ontologies. Journal of Biomedical Semantics **6**, 12:1–26 (2015). https://doi.org/10.1186/s13326-015-0002-8
4. Musen, M.A.: The protégé project: a look back and a look forward. AI Matters **1**(4), 4–12 (2015). https://doi.org/10.1145/2757001.2757003, `https://doi.org/10.1145/2757001.2757003`
5. Pesquita, C., Faria, D., Santos, E., Couto, F.M.: To repair or not to repair: reconciling correctness and coherence in ontology reference alignments. In: Shvaiko, P., Euzenat, J., Srinivas, K., Mao, M., Jiménez-Ruiz, E. (eds.) Proceedings of the 8th International Workshop on Ontology Matching. CEUR Workshop Proceedings, vol. 1111, pp. 13–24 (2013), `http://ceur-ws.org/Vol-1111/om2013_Tpaper2.pdf`
6. Pesquita, C., Ivanova, V., Lohmann, S., Lambrix, P.: A framework to conduct and report on empirical user studies in semantic web contexts. In: Faron-Zucker, C., Ghidini, C., Napoli, A., Toussaint, Y. (eds.) Knowledge Engineering and Knowledge Management - 21st International Conference, EKAW 2018, Nancy, France, November 12-16, 2018, Proceedings. LNCS, vol. 11313, pp. 567–583 (2018). https://doi.org/10.1007/978-3-030-03667-6_36

KG Explorer: a Customisable Exploration Tool for Knowledge Graphs

Thibault Ehrhart, Pasquale Lisena, and Raphaël Troncy

EURECOM, Sophia Antipolis, France
{ehrhart,lisena,troncy}@eurecom.fr

Abstract. The growing adoption of Knowledge Graphs demands new applications which enable users to search and browse structured data in a suitable way depending on the domain. In this paper, we introduce KG Explorer, a web-based exploratory search engine for RDF-based Knowledge Graphs. The software can be configured in order to adapt to different information domains, customising both the UI components and the queries made for retrieving the information. It also includes features such as full-text search, facet-based advanced search, and the possibility to create lists of favourites items modelled in the knowledge graph.

Keywords: knowledge graphs, data exploration, data access, search interface

1 Introduction

Knowledge Graphs (KG) are more and more adopted for representing the information: today we can find several graphs, small and large, which may represent encyclopedic-general or domain-specific information. Their still growing popularity is due to an interesting set of characteristics, such as explicit semantic, interlinking with external resources, and a great expressiveness coming from Semantic Web technologies. KGs offer structured data that empower semantic search and QA systems among many other possible applications.

More recently, we see the interest in creating beautiful visualisation of KG-powered search results. An example is the use of Knowledge Panels on Search Engines, which are currently moving from simply displaying key-value tuples to integrate images and text for presenting the information nicely (Fig. 1).

Knoweldge Graphs can be stored in dedicated triple store. Those, generally offer – next to the essential SPARQL endpoint – a browsing user interface (UI), which allows an end-user to see the loaded data on a web page. For example, the *facet browser* of Virtuoso[1] shows all incoming and outgoing predicates for a given resource with the respective values[2]. When the value represents a picture, the image is retrieved and displayed in the page. All entity nodes and edges are clickable, so that the user can navigate

[1] http://vos.openlinksw.com/

[2] Example from DBpedia: https://bit.ly/3z5nPLG

Fig. 1. The Knowledge Panel for the search keyword "Goat" in Google (left) and Bing (right). Screenshot taken on 19/03/2021

through the graph in a *follow-your-nose* approach. Other common features are plain text search (`/fct` on Virtuoso), query helper (YASGUI[3]), dereferencing service for linked data URI, or rich visualisation of results (like in Wikidata[4]).

However, these systems fall short when it is necessary to go beyond the simple visualisation of text and images and:

- embrace different media objects, such as video, audio, 3D graphics;
- propose new navigation paradigms, such as related items or recommendations for the next element;
- improve the search and exploration experience based on the domain peculiarities, filtering the results based on time ranges, geographic areas, or values form hierarchical thesauri. Moreover, the connection of the searched object and the value to filter can consist of a single direct property, a property path, or even a more complex query.

To provide a generic solution to these limitations, we introduce KG Explorer, a fully-customisable web application which serves as exploratory search engine [16] for Knowledge Graphs. KG Explorer offers alternative ways to browse a graph, to search and to follow links, to discover new information by exploiting the semantic proximity of entities. Respect to other works in literature, KG Explorer gives the possibility to configure the application components, that can be included or excluded and customised in the functionality or in the visualisation. In addition, a *save in the list* feature is included, bringing to information discovery applications a pattern coming from shopping exploratory search engine.

[3] `https://triply.cc/docs/yasgui-api`
[4] See for example `https://www.wikidata.org/wiki/Q2934`

This paper is organised as follows. After discussing some related work in Section 2, we detail some shared needs for the application in Section 3. The functionalities and architecture of KG Explorer are described respectively in Section 4 and 5. We present a preliminary evaluation in Section 6 and some conclusion in Section 7.

2 Related Work

An extensive survey of facet search has been published in [21]. This work has the merit of defining the basic concepts for the exploratory approach, namely the *extension* (the displayed results), the *intension* (the satisfied query) and the *transition markers*, clickable elements for triggering a transition (a new query). In addition, the work points out the possible kind of configuration of the tool, from the absence of any configuration requirement to the exact content to be displayed (view-based configuration).

Faceted Wikipedia Search [7] is a facet search tool based on DBpedia. The transition markers are sorted and displayed based on their frequency with respect to the number of results, in order to help the user in refining her/his query in successive iterations. Other provided features are free text search and range selection for datatype values. A similar interaction is implemented in *GraFa* [15], which refines the facet list after selecting the text keyword to search or the desired entity type. The involved schemas are indexed in order to have quick response, applying in addition a materialisation for the query returning the bigger number of results. These solutions are, however, based on statistics computed on properties, and do not take into account the domain specificity. In fact, the chosen facets are not always relevant nor useful for the search experience. The Metaphacts ecosystem[5] includes an extension for building customisable apps on top of Linked Data.

FERASAT [9] shows the results obtained through combination of facet values in different visualisation components (maps, charts, etc.), in order to make evident the surprising results. This application targets a public of data experts but it would be quite complex for a broader audience. *LDVizWiz* [1] provides aggregate visualisations for entities of specific types in a KG, such as events which can be displayed on maps, timelines and tables. *Loupe* [14] displays the ontology classes and properties frequently used in tabular format, allowing the user to see how they are normally combined in the triples. These works show exclusively aggregate results, without enabling any customisation depending on the investigated domain.

In *Overture* [11], the visualisation of entity data is extended with custom components, showing a timeline of relevant events and the most similar entities from on a knowledge-based recommender system. In the *WarSampo portal* [10] (about Finnish history in World War II), different tabs allows to switch between a tabular visualisation of data, a timeline and a map, and a photo gallery[6]. The resource page of *Genesis* [5] includes entity textual data, images and videos, as well as a selection of similar and related entities with their own depictions. These examples are ad-hoc developed tools, hard to adapt to new domains.

[5] `https://metaphacts.com/`

[6] Example: `https://www.sotasampo.fi/en/persons/person_61`

The Fresnel vocabulary [17] has been proposed for closing the gap between data and presentation, enabling to define content subsets and formats matched with CSS classes. Similarly, custom views are used for driving the visualisation in [2,20,4]. However, these approaches do not propose solutions to data search. Works like *PepeSearch* revealed good search capabilities, but the results are only shown in tabular form [22]. Other works combines exploratory search with facets [13,23]; however, these works do not focus on customising the user interface.

3 Different Scenarios But Shared Needs

Different users may take advantage from data inside specialised Knowledge Graphs, each one with their own needs and goals. We identified the following shared needs:

- to understand what is in the dataset, and in particular the main resource types (classes) and how they are connected to each other;
- to search for specific resource which satisfy some domain-relevant criteria;
- to obtain detailed information about a particular resource, including multimedia data and smart aggregations using timelines, maps and plots.

These need are highly impacted by the kind of user, which can fall in one of the following scenarios:

- *domain experts* have great interest in the subject, are used to the domain vocabulary and know what they search with precision. They need advanced search capabilities, allowing them to filter the results by several dimensions. The information needs to be complete.
- the *wide public* is rather moved by the curiosity of discovering something new, sometimes having only general or null knowledge about the domain. They need to easily browse the data collection and possibly reach relevant information already after the first click. Some strategies are needed to make them continue the exploration, for example follow-your-nose approaches or the recommendation of similar or related items. The engagement is crucial for their experience.
- *external stakeholders* need to know which relevant information is possible to find in the data and how to easily access it.

We argue that an exploratory search engine [16] enables to fulfil the described needs while being flexible enough to targeting the different personas. In addition, the application should have a proper user interface (UI), which reflects the domain specificity and the institution identity. In the same time, this can improve the final user engagement. Further desired elements are the selection of the language for KGs including multi-lingual contents and an authentication method for data that are not public.

4 KG Explorer Functionalities

Having defined the final goals, we are going to detail in this section the features implemented in KG Explorer: a facet-based advanced search engine, dedicated editorial pages for controlled vocabularies represented in SKOS and generally used in the

knowledge graph, a customised detailed page for the main entities represented in the knowledge graph, the possibility for users to log in and to create personalised lists of favourites or saved items. The software can be configured to adapt to different information domains, changing not only its aspect but also the queries for retrieving the data to display. KG Explorer is open source under Apache License 2.0 at `https://github.com/D2KLab/explorer`. In order to explain the software capabilities, we will refer to three in-use applications of KG Explorer. These examples use data coming from different domains (cultural heritage, television and news), each of them with proper customisation. The links to the applications and the source code are collected in Table 1.

ontologies	#entities	links
ADASilk - *domain*: silk heritage		
CIDOC-CRM CRMsci	675,112	Source code: `https://git.io/adasilk` Application: `https://ada.silknow.org/`
MeMAD Explorer - *domain:* TV and Radio programmes		
EBUcore	1,079,969	Source code: `https://git.io/memad-explorer` Application: `https://explorer.memad.eu/`
ASRAEL Search Engine - *domain:* news and events		
OpenAnnotation rNews schema.org	968,602	Source code: `http://bit.ly/asrael-se` Application: `http://asrael.eurecom.fr/search-engine`

Table 1. In-use instances of KG Explorer (including ASRAEL Search Engine which is a fork of the main tool).

4.1 A standardised experience

KG Explorer offers a user experience based on four different kinds of pages. The **landing page** contains a search box which allows the user to perform a free text search on entities modeled in the Knowledge Graph. When the user enters a search term, the exploratory search engine executes a SPARQL query with a `REGEX` filter in order to select all items that have a label or a title that partially matches the search terms. The search query algorithm can also be changed in the configuration file to cover all datatype properties of the graph. The results are shown in an auto-complete box.

The **browse page** (Fig. 2) contains a faceted search engine which allows users to perform an advanced search for the main entities of the Knowledge Graph. The sidebar on the left side contains facets (or filters). Each facet generates an extra condition to the main SPARQL query used for searching.

In addition to a textual search box, the exploratory search engine provides shortcuts to so-called **vocabulary pages**, which show all terms belonging to a particular thesaurus – e.g. a *ConceptScheme* in the SKOS namespace. These vocabularies are defined in the configuration file, and are usually materialised as concepts in the KG. Clicking on a vocabulary term will bring the user to a pre-filtered browse page, in order to see the related items in the graph.

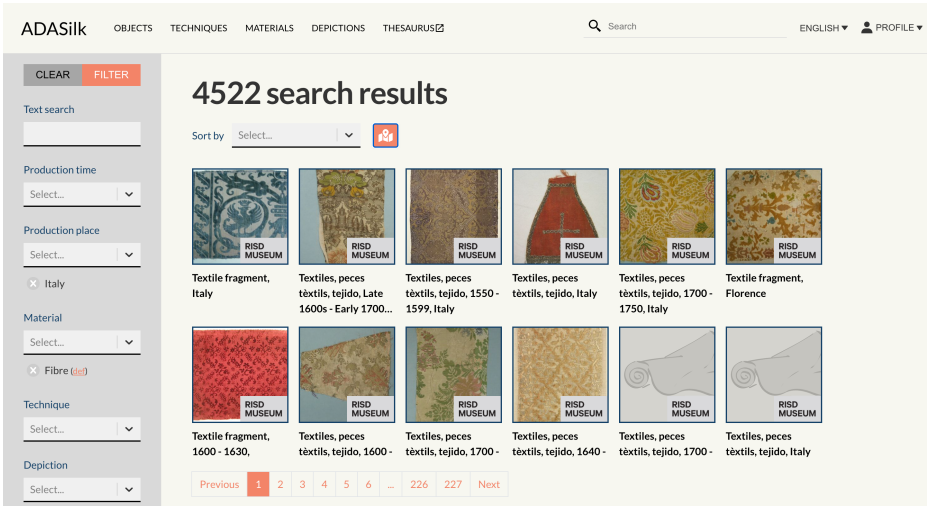


Fig. 2. The browse page in ADASilk.

Finally, the **detail page** shows all the information related to a single entity. There are currently 3 layouts available for detail pages: collection (grid-based list of items), gallery (carousel of images), and video (media player). Custom pages can be added by creating new JavaScript files in the `pages/` directory, and exporting the class as a React component. Each page is automatically included in the build and associated with a route based on its file name. New layouts can also be added to the project, by creating a new file in the `pages/details/` directory, and referring to its name in the `view` property in the configuration file. This is being used for developing the video player view in the MeMAD Explorer, handling also authentication to the media server.

4.2 User profiles

KG Explorer includes an authentication system which allows users to create an account, log in and have access to additional features. The OAuth authentication method is used for creating a new profile and for any successive login, relying on signing-in via Google, Facebook, and Twitter. Once logged in, users have the possibility to create named lists for storing searched items. A “save” button is present on each detail page, allowing to add the current page to an existing list or to create a new one. Lists can be retrieved in the profile page of the user, from where they can also be made public and shared with anyone using a permalink. Moreover, from the profile page it is possible to link or unlink additional OAuth accounts, as well as manage the existing lists or even delete the user profile.

4.3 Generic tool, custom configuration

Each domain and KG has its own characteristic. KG explorer is capable of working on top of any RDF-based Knowledge Graph, by configuring an instance of it using a

JavaScript file (`config.js`). The configuration allows to define a wide set of options, such as the chosen SPARQL endpoint, the supported language for internationalisation, and some layout-related settings – i.e. which images to use, which components to show or hide, etc.

Of particular interest is the possibility of defining the pages that compose the application, through the `route` field of the configuration file. The example in Listing 1[7] shows the available options, which include the choice between *browse* or *vocabulary* page, the page URI, the applied JSON query for listing the results (following the SPARQL Transformer syntax, as described in Section 5).

In *browse* pages, the `filters` property can contain a list of available fields for the advanced search, detailing also which changes are applied to the query when filters are applied. The list of available values can be loaded with a query (defined or made globally available as *vocabulary*). The main query condition is defined with the *baseWhere* property, with the minimal amount of triples required in order to improve performances. Once the list of results has been fetched, a second query is made to get the details of each result. This query is defined within the *query* property. The labels for the **internationalisation** are collected in specific JSON files to include in the project directory.

The front-end also supports **custom styles** which can be defined in a `theme.js` file. This allows to further customise the appearance of the user interface. It is possible to choose the global font set and a custom colour palette. Moreover, specific components can also be customised, by using the name of component and defining CSS rules following the *styled-components* syntax. Finally, adding custom pages and view (Section 4.1) enable the developer to include new **visualisation components**. Examples are maps and 3D visualisation in ADASilk.

5 Architecture

Fig. 3 shows an overview of the architecture and the technologies used in KG Explorer. KG Explorer is developed in a **containerised approach**, implemented within the Docker framework[8]: thanks to the use of independent and self-sufficient containers, Docker enables the deployment of this architecture on any machine, automatically installing and running the required software. This approach also allows to easily extend and deploy new instances of the application from the base image, including custom configuration and assets, as has been done in the instances in Table 1.

The **web application** is composed of several web technologies. The front-end is produced using *React*[9]. It uses encapsulated components that manage their own state to help maximise code re-usability. *Next.js*[10] is used for server-side rendering and page-based routing. It relies on a file-based structure for routing, where each page has its own file, stored in the `src/pages` directory. Special routes are dedicated to serve

[7] The code is extracted from the ADASilk configuration and is fully available at `https://github.com/silknow/adasilk/blob/main/config/routes/object.js`

[8] `https://www.docker.com/`

[9] `https://reactjs.org/`

[10] `https://nextjs.org/`

```
{
    objects: {
        view: 'browse', // type of view ('browse' or 'vocabulary')
        showInNavbar: true,
        rdfType: 'http://erlangen-crm.org/current/E22_Man-Made_Object',
        uriBase: 'http://data.silknow.org/object',
        details: { view: 'gallery' },
        filters: [{  // set of filters to appear in the advanced search
          id: 'material', // material filter
          isMulti: true, // 1 or more values can be selected
          isSortable: true,
          vocabulary: 'material', // values taken from a vocabulary
          whereFunc: () => [ // added to the base query when filtering
            '?production ecrm:P126_employed ?material',
            `OPTIONAL {
               ?broaderMat (skos:member|skos:narrower)* ?material }`
          ],
          filterFunc: (values) => { // add to base query when filtering
            return [values.map((val) =>
            `?material = <${val}> || ?broaderMaterial = <${val}>`)
                .join(' || ')];}
        }],
        baseWhere: [
            'GRAPH ?g { ?id a ecrm:E22_Man-Made_Object }',
            '?production ecrm:P108_has_produced ?id',
        ],
        query: { // base query
          '@graph': [{
            '@type': 'http://erlangen-crm.org/.../E22_Man-Made_Object',
            '@id': '?id',
            '@graph': '?g',
            label: '$rdfs:label',
            identifier: '$dc:identifier',
            description: '$ecrm:P3_has_note',
          }],
          $where: ['GRAPH ?g { ?id a ecrm:E22_Man-Made_Object }']
        }
    }
}
```

Listing 1: Partial definition of the ‘Objects’ route in ADASilk, with the optional filter by material

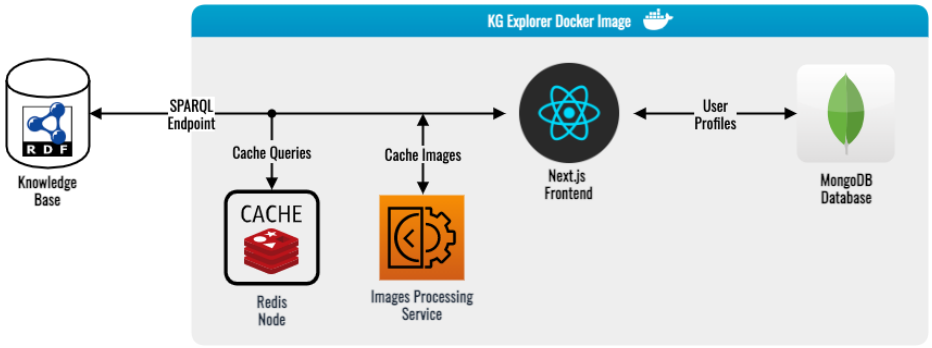


Fig. 3. Architecture of KG Explorer

APIs, used on the server-side for handling authentication, fetching profile data, and searching. The library *styled-components*[11] is used for styling components using scoped CSS. Other used frameworks are *i18next*[12] for the internationalisation and *next-auth*[13] for OAuth authentication.

KG Explorer makes requests to a Knowledge Graph through its exposed SPARQL endpoint. In order to easily include and manipulate queries in JavaScript, those are written in the JSON query syntax proposed by **SPARQL Transformer** [12]. The SPARQL Transformer library makes it easy to define queries using JavaScript objects (called *JSON queries*) which can be edited and merged to create the final query. For instance, each filter from the faceted search appends its own conditions to the base query, as seen in Section 4.3. Looking again at Listing 1, when a filter is applied, the base query is modified applying new `WHERE` and `FILTER` expressions, respectively defined in `whereFunc` and `filterFunc`. The use of JSON queries makes it possible to simply append this expression in the `$where` and `$filter` properties of SPARQL Transformer, and avoids a much more complex manipulation of text which the use of plain SPARQL queries would require. SPARQL Transformer also rewrites the output of SPARQL queries in a more suitable format for web development. In particular, SPARQL results composed of bindings between variables and solutions are transformed into self-contained JSON objects, including all the information about the entities, getting rid of some verbosity of the standard notation. Queries results are processed and cached into a *Redis* database[14] in order to improve performances. The results are stored as a JSON string, and the original query is used as the key for retrieving the cached result. User profiles and lists are saved in a *MongoDB* database[15].

[11] `https://styled-components.com/`
[12] `https://www.i18next.com/`
[13] `https://next-auth.js.org/`
[14] `https://redis.io/`
[15] `https://www.mongodb.com/`

6 Preliminary Evaluation

Preliminary evaluations of KG Explorer were conducted as part of the SILKNOW project [18]. The application has been used by 216 users, reflecting different audience, domain and technical skills (Table 2). The users were asked to perform some search activities and to comment on the results reflecting both the intrinsic quality of the knowledge graph which is hard to isolate and the ability of searching for specific items and of browsing and discovering new items.

Domain	English	French	Spanish	Italian	**Total**
Cultural Heritage	0	0	14	14	**28**
Education related to social science	1	0	6	4	**10**
Information and communication technology	1	17	42	67	**126**
Textile or creative industry	0	1	1	1	**3**
Tourism	0	1	0	2	**3**
Media	0	2	2	3	**7**
Other	0	2	15	22	**39**
					216

Table 2. Target audience used during the evaluation of ADASilk.

During the evaluation, each user session has been recorded, after consent, for successive analysis. To do this, the rrweb[16] library is implemented into the UI in order to record and then replay each interaction with the interface. The recorded sessions are saved as JSON objects in a database. At the end of the evaluation, the sessions were exported as MP4 videos using rrvideo.[17] We report below the most common issues and what users perceive as anomalous behaviour.

From the analysis of all the tests conducted through ADASilk, a commonly encountered issue is related to the text search functionality. While offering free text search was found to be an essential feature, it also raises some expectations that the search query will be somehow interpreted. Users are familiar with Google which interprets and disambiguates search queries while offering personalized answers. In contrast, KG Explorer offers either a naive *text search* that aims to match resources for which the search terms can be encountered in a datatype property value or a *concept search* which can lookup and auto-complete concepts from controlled vocabularies typically used in facets. Often, users have entered simple search strings expecting that their translations in other languages will bring the same result set.

The relevance of the search results was also pointed out as an issue during the evaluation, in particular, by domain experts. The sole SPARQL query language offers only the possibility of returning a set of exact solutions to a query without natural ways of ranking the resources within this set nor with the possibility to consider partially related resources. The numerous methods enabling to build knowledge graph embeddings are

[16] `https://github.com/rrweb-io/rrweb`
[17] `https://github.com/rrweb-io/rrvideo`

promising to bring this notion of relevance, e.g., in measuring the distance between each document. We observe that some triple stores, such as GraphDB[18], have started to provide native support for semantic similarity searches.

After a software improvement in order to overcome the observed limitations, a second evaluation has been done using the System Usability Score (SUS) questionnaire [3]. The participants has been composed of 125 people representative of the previously defined stakeholders. The participant were speaking English, Spanish or Italian, were mostly of higher education (75%) and in the age range 21-30 (59%)[19]. Even with possibility of improvement – the system obtained a SUS score of 67.03 – over 70% of the participants declared the intention to use it [6].

7 Conclusion and Future Work

KG Explorer provides a domain-specific user experience for exploring the information contained in a Knowledge Graph. The software can be easily customised and adapted in the UI and in the content, defining the queries for retrieving the data, the facets to be used, and the relevant vocabularies. KG Explorer is already used in real-world applications, in particular as wide-public entry-point for Knowledge Graphs of research projects. In this context, a user evaluation is currently being carried out where the goal is to measure the usability of the application in the fulfilment of common tasks, identified by domain experts. The outcome of this evaluation will be used for further improving the application.

Future developments will also involve new functionalities such as having custom facet selectors for datatypes, for example ranges for numbers and dates. Finally, we would like to exploit the vocabularies in order to provide a smart text search field, going beyond the exact match on text: this can be implemented by recognising terms defined in vocabularies and attaching them to the most appropriate property in the generated query, in a query interpretation behaviour. In this field, previous research has proved the suitability of embedding techniques for representing a query, in order to get more relevant results [8,24]. This text search may be also be further combined with structured search.

Acknowledgements

This work has been partially supported by the European Union's Horizon 2020 research and innovation program within the SILKNOW (grant agreement No. 769504) and MeMAD (grant agreement No. 780069) projects, and by the French National Research Agency (ANR) within the ASRAEL project (grant number ANR-15-CE23-0018).

References

1. Atemezing, G.A., Troncy, R.: Towards a Linked-Data Based Visualization Wizard. In: 5^{th} International Conference on Consuming Linked Data (COLD). Riva del Garda, Italy (2014)

[18] `https://graphdb.ontotext.com/`

[19] More detail about distribution of participants is available in [19]

2. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and Analyzing linked data on the Semantic Web. In: 3^{rd} International Semantic Web User Interaction Workshop (SWUI) (2006)
3. Brooke, J.: SUS: a retrospective. Journal of usability studies **8**(2), 29–40 (2013)
4. Chauvat, N., Amarger, F., Wouters, L.: Un navigateur pour le Web des données liées. In: 30es Journées Francophones d'Ingénierie des Connaissances, IC 2019. pp. 167–182. Toulouse, France (2019)
5. Ermilov, T., Moussallem, D., Usbeck, R., Ngonga Ngomo, A.C.: GENESIS: A Generic RDF Data Access Interface. In: International Conference on Web Intelligence (WI). pp. 125—131 (2017)
6. Gaitán, M., León, A.: SILKNOW System Evaluation. project deliverable D7.6, H2020 SILKNOW (2021)
7. Hahn, R., Bizer, C., Sahnwaldt, C., Herta, C., Robinson, S., Bürgle, M., Düwiger, H., Scheel, U.: Faceted Wikipedia Search. In: 13^{th} Conference on Business Information Systems (BIS) (2010)
8. Hamilton, W.L., Bajaj, P., Zitnik, M., Jurafsky, D., Leskovec, J.: Embedding Logical Queries on Knowledge Graphs. In: 32^{nd} International Conference on Neural Information Processing Systems (NIPS). pp. 2030—2041 (2018)
9. Khalili, A., van den Besselaar, P., de Graaf, K.A.: FERASAT: A Serendipity-Fostering Faceted Browser for Linked Data. In: 17^{th} International Semantic Web Conference (ISWC). pp. 351–366 (2018)
10. Koho, M., Ikkala, E., Leskinen, P., Tamper, M., Tuominen, J., Hyvönen, E.: WarSampo knowledge graph: Finland in the Second World War as Linked Open Data. Semantic Web Journal pp. 1–14 (2020)
11. Lisena, P., Achichi, M., Fernandez, E., Todorov, K., Troncy, R.: Exploring Linked Classical Music Catalogs with OVERTURE. In: 15^{th} International Semantic Web Conference (ISWC), Posters & Demos Track. Kobe, Japan (2016)
12. Lisena, P., Meroño-Peñuela, A., Kuhn, T., Troncy, R.: Easy Web API Development with SPARQL Transformer. In: 18^{th} International Semantic Web Conference (ISWC). pp. 454–470. Auckland, New Zealand (2019)
13. Marie, N., Gandon, F., Ribière, M., Rodio, F.: Discovery Hub: On-the-Fly Linked Data Exploratory Search. In: Proceedings of the 9th International Conference on Semantic Systems. p. 17–24. I-SEMANTICS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2506182.2506185, https://doi.org/10.1145/2506182.2506185
14. Mihindukulasooriya, N., Poveda-Villalón, M., García-Castro, R., Gómez-Pérez, A.: Loupe - An Online Tool for Inspecting Datasets in the Linked Data Cloud. In: 14^{th} International Semantic Web Conference (Posters & Demos) (2015)
15. Moreno-Vega, J., Hogan, A.: Grafa: Scalable faceted browsing for rdf graphs. In: 17^{th} International Semantic Web Conference (ISWC). pp. 301–317 (2018)
16. Palagi, E., Gandon, F., Giboin, A., Troncy, R.: A Survey of Definitions and Models of Exploratory Search. In: ACM Workshop on Exploratory Search and Interactive Data Analytics (ESIDA). Limassol, Cyprus (2017)
17. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In: 5^{th} International Semantic Web Conference (ISWC). pp. 158–171 (2006)
18. Seidita, V., Lo Cicero, G., Vitella, M.: Testing Report in a Real Scenario. project deliverable D7.2, H2020 SILKNOW (2021)
19. Seidita, V., Lo Cicero, G., Vitella, M., Mladenic, D., Gaitán, M., Troncy, R., Portales, C.: Usability evaluation by online users of the system. project deliverable D7.3, H2020 SILKNOW (2021)

20. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: Sig.ma: Live views on the Web of Data. Journal of Web Semantics **8**(4), 355–364 (2010)
21. Tzitzikas, Y., Manolis, N., Papadakos, P.: Faceted Exploration of RDF/S Datasets: A Survey. Journal of Intelligent Information Systems **48**(2), 329—-364 (2017)
22. Vega-Gorgojo, G., Giese, M., Heggestøyl, S., Soylu, A., Waaler, A.: Pepe-search: Semantic data for the masses. PLOS ONE **11**(3), 1–12 (03 2016). https://doi.org/10.1371/journal.pone.0151573, https://doi.org/10.1371/journal.pone.0151573
23. Waitelonis, J., Sack, H.: Towards exploratory video search using linked data. Multimedia Tools and Applications **59**(2), 645–672 (Jul 2012). https://doi.org/10.1007/s11042-011-0733-1, https://doi.org/10.1007/s11042-011-0733-1
24. Xiong, C., Power, R., Callan, J.: Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In: 26^{th} International Conference on World Wide Web (WWW). pp. 1271—-1279. Perth, Australia (2017)

Timelining Knowledge Graphs in the Browser

Damien Graux[(✉)1,2], Fabrizio Orlandi[(✉)2],
Tanmay Kaushik[2], David Kavanagh[2], Hailing Jiang[2], Brian Bredican[2],
Matthew Grouse[2], and Dáithí Geary[2]

[1] Inria, Université Côte d'Azur, CNRS, I3S, France
[2] ADAPT SFI Research Centre & Trinity College Dublin, Ireland
{grauxd,orlandif}@tcd.ie

Abstract. Knowledge graphs, available on the Web via SPARQL endpoints, provide practitioners with various kinds of information from general considerations to more specific ones such as temporal data. In this article, we propose a light-weight solution to visually grasp, navigate and compare, in a Web browser, temporal information available from SPARQL endpoints. Furthermore, we use Wikidata's public SPARQL endpoint to demonstrate our solution and allow users to navigate Wikidata's temporal information.

1 Introduction

Over the past two decades many data sources have been published on the Web. Most of the time, they follow the recommendations and standards promoted by the World Wide Web Consortium (W3C) within the Semantic Web movement, driven by the desire to create a "Web of data" from the conventional "Web of documents". These datasets, generally represented thanks to the RDF format [9] and accessible via the SPARQL language [12], deal with subjects ranging from generalist knowledge such as DBpedia [8], YAGO [10] or Wikidata [11] to specific knowledge such as legal court cases [6], source codes [7] or medical information [13]. Thus, the amount of semantic data now (publicly) accessible makes it possible to create new applications combining for instance several datasets at once. In addition, among the nowadays available datasets, several ones are open online and offer public endpoints on which users may send queries.

To help users navigate this large amount of RDF information, data architects rely on the use of ontologies to structure their datasets. Typically, they declare how entities may be related to each other, for instance a person is of type *human* and should have *parents* who are also of type *human*[1]. More specifically, any kind of type and or relation could be represented in RDF and thus serves to structure knowledge. Among the various facets which might be represented, one is often present in knowledge graphs: temporal information. These can take

[1] See for example the friend-of-a-friend structure devoted to linking people and information using the Web: http://xmlns.com/foaf/spec/

various forms, from the representation of a specific point in time (*i.e.* a date) to a time span. Practically, in RDF, dated information can correspond to a typed date represented by a literal like so: `"25/10/2021"^^xsd:date` which means that objects are the date-*carriers*[2] and predicates express is the date is related to a point in time, or to a starting date, *etc.* As a consequence, there is available temporal information in most of the RDF knowledge graphs, however, having access to it is not a trivial task as the RDF triple structure splits the information across several statements.

In this article, we present an approach to gather, in a Web browser, temporal information coming from SPARQL endpoints. We describe how our system relies on SPARQL queries in order to retrieve the necessary information and how the pipeline can be modified to adapt to other data sources. Visually, our solution allows to present temporal information using timelines which are enriched with additional features such as "click-&-follow" or "compare" nodes. Finally, to validate our approach, we deploy our strategy on the Wikidata [11] public endpoint and host it on a Github page at https://wikitimeline.github.io/.

2 Collection of temporal information

As introduced in the previous section, knowledge graphs are generally structured using the RDF standard [9]. Technically it organises data in *triple* statements composed of a *subject*, a *predicate* and an *object*. In addition to that, each element is entitled to have a specific semantic role: the predicate can only be a Uniform Resource Identifier (*e.g.* <http://purl.org/dc/terms/title>[3]) when an object can also be a literal value such as `"some description"@en` tagged by the English language. Therefore, if there are some temporally typed data existing in an RDF dataset, it should be as a literal in an object field.

Associated with the RDF standard comes SPARQL [12]: its *de facto* query language, standardised by the W3C too. SPARQL adopts an imperative SQL-like syntax for fetching information together with graph-specific features and a large set of functions specific to dealing with RDF data; for example, `isIRI` checks whether the mapped value behind a variable is a URI or not, and it can be used to test a variable having an object role in the graph...

As a consequence, using exclusively SPARQL (standard) queries, we can filter the graphs in order to retrieve temporally typed/tagged information together with their related context (close-triples). For example in:

```
SELECT * WHERE {
 ?s ?p ?o . FILTER ( isLiteral(?o) )
 FILTER ( DATATYPE(?o) = <http://www.w3.org/2001/XMLSchema#date> )
}
```

The query will run through the entire RDF default graph (pattern `?s ?p ?o`) and returns the triple if the object `?o` is a literal **and** if it is a date `<..Schema#date>`.

[2] As only objects can be literal in RDF [9].
[3] URIs are *dereferenceable* and could thereby be mapped to a resource.

In the context of more complex / richer knowledge graphs, we apply a similar strategy following the dataset ontology in order to know how (and where) the dates or the temporal information are stored.

3 Timelines in the Web browser

In order to be able to visualise temporal information from triple endpoints, we devised a twofold architecture. First, we design a way to render such information visually and thus decide to represent time spans using timelines. In particular, we choose to rely on the timelines-chart library[4] to draw and navigate through temporal information. Practically, the solution is implemented using basic JavaScript and does not require the use of heavy external libraries. Second, to fetch the necessary data needed by the visualiser, we write SPARQL queries to be sent to the chosen endpoint where the RDF triples are stored. These queries are in charge of filtering the graphs to extract only the "interesting" parts together with the temporal information.

3.1 Adaptability with SPARQL

For adaptability purposes, we design our approach so that changing the accessed SPARQL endpoint (and thus the RDF graphs queried) does not imply a complete redevelopment of our tool. Indeed, to change the considered knowledge graph, one should modify (1) the address of the endpoint and (2) the queries used. If the first step is straightforward, the second may still lead to some simple query rewriting. To limit this effort, we compact the queries so that all the necessary fields are retrieved with two queries. More generally, these two queries should return the following elements to guarantee that the visualiser can properly process the data:

Q_1: `SELECT ?pred ?obj ?pName` – to find all dates (`?obj`) and their relationships (`?pred & ?pName`) related to a specified subject;
Q_2: `SELECT ?pred ?pName ?obj ?oName ?start ?end` – to find start & end dates metadata about statements (`?sub ?pred ?obj`) on a specified subject.

Such a uniform query output structure allows therefore the practitioners to change the content of the query as much as they want. For instance, on Wikidata [11], and considering the "*qualifiers*"[5] *i.e.* the way Wikidata represents statements of statements, Q_2 would be as follows:

```
SELECT ?pred ?pName ?obj ?oName ?start ?end WHERE {
 ?subj ?pred ?statement. ?statement ?predPS ?obj.
 ?statement pq:P580 ?start.
 OPTIONAL { ?statement pq:P582 ?end. FILTER(?end >= ?start).}
 FILTER(STRSTARTS(STR(?predPS), "http://www.wikidata.org/prop/statement/"))
 FILTER(STRSTARTS(STR(?pred), "http://www.wikidata.org/prop/"))
 FILTER(STRSTARTS(STR(?statement), "http://www.wikidata.org/entity/statement/"))
 ?x wikibase:claim ?pred. ?x rdfs:label ?pName.
```

[4] https://github.com/vasturiano/timelines-chart (built on D3.js)
[5] https://www.wikidata.org/wiki/Help:Qualifiers

```
 ?obj rdfs:label ?oName.
 FILTER((LANG(?pName)) = "en"). FILTER((LANG(?oName)) = "en").
}
```

However, on YAGO4 [10], considering that RDF-star [5] is used to reify triples, Q_2 would be:

```
SELECT ?pred ?pName ?obj ?oName ?start ?end WHERE {
 << ?subj ?pred ?obj >> <http://schema.org/startDate> ?start .
 OPTIONAL{ << ?subj ?pred ?obj >> <http://schema.org/endDate> ?end .}
 ?pred rdfs:label ?pName .
 ?obj rdfs:label ?oName .  FILTER(LANG(?oName)='en').
}
```

Hence, our solution can be deployed on any endpoint as long as the involved queries are answering the same structures.

3.2 Visual Features

From the graphical point-of-view, we develop several features to enrich the users' experience. Practically, timelines are representing temporal information related to one single entity (usually a subject in RDF). As the number of elements describing a specific entity can be large and since time can span over decades for some entity (such as states for instance), the interface enables users to zoom on a particular period of time in order to "restrict" the window-view, using a *click-and-drag* movement. Similarly, to help users distinguishing similar kinds of information, we group the same predicates by color. Additionally, more details can be read when the users *hover* above a time-bar displaying the exact starting and ending dates of it. Furthermore, in order to "navigate within" the endpoint's RDF graph, we set up a *click-and-follow* mechanism on the time-bars. Actually, clicking on time-bar redirects the users on the timeline corresponding to the entity the initial bar was about. Such a mechanism allows a seamless experience for the user who can explore the available temporal elements from one entity to another. Finally, we implemented a specific *compare* feature to allow users comparing two entities at once by grouping the two timelines together.

3.3 Practical use-case: Wikidata timelines

In order to validate our approach, we tested the system using the available Wikidata SPARQL endpoint. To do so, we designed the necessary SPARQL queries and hosted our interface online. Typically, a comparison between two entities is shown in Figure 1 where the timelines of Presidents George W. Bush and Barack Obama are displayed. In addition, for this online interface, we added an auto-completion feature to fasten the search of any entity as Wikidata uses exotic names *e.g.* B. Obama is "Q76". Finally, following on the dereferenceable principle, we also enable URL access to let user share timelines, for example Tim Berners-Lee's timeline is available from: https://wikitimeline.github.io/search.html?subj=http://www.wikidata.org/entity/Q80.

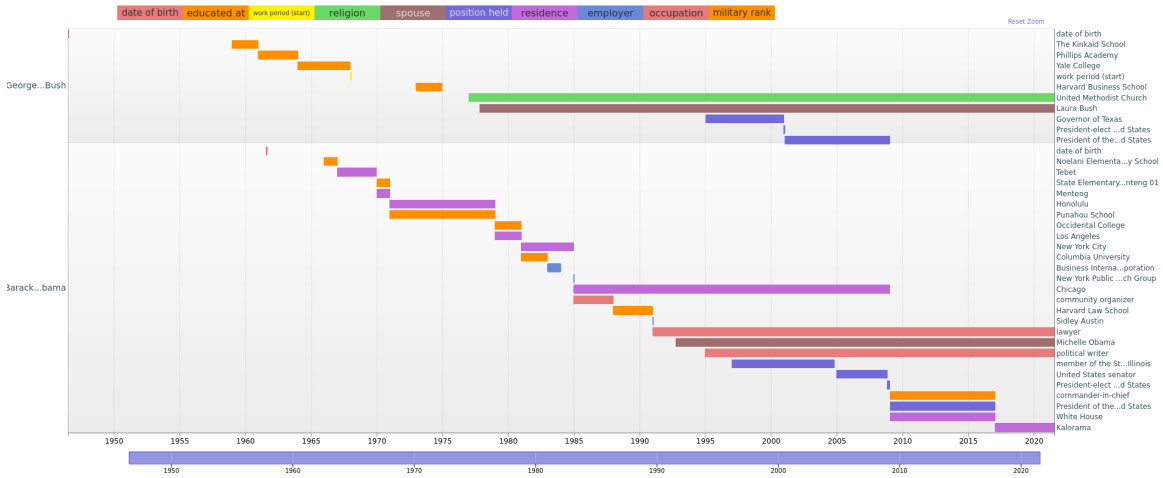


Fig. 1. Comparing George W. Bush and Barack Obama. See these timelines

4 Related Work

To the best of our knowledge, the literature does not contain solutions focusing exclusively on "timelining" temporal information contained in KGs. Nevertheless, timeline representations have already been used within larger platforms as a side-tool such as Metaphactory [4]. Similarly, `rdf:SynopsViz` [1] has a dedicated tab to draw timeline from date properties available in RDF datasets; however, the platform needs to prior ingest the dataset, whereas our solution relies on SPARQL endpoints. Recently, Gottschalk et al. [2,3] formalised the notion of temporal knowledge graph and instantiated it using several bases such as YAGO or Wikidata. Regarding the visualisation, the difference with our approach is that they created their own KG, based on their own schema, therefore the visualisations need to be updated when original datasets change.

5 Conclusion

The proposed web-app visualises and compares Wikidata entities according to their temporal information. A demonstrator is hosted on:

https://wikitimeline.github.io/

under an MIT license[6], providing users a live example of what the application could be locally, would someone be interested in deploying the interfaces at their premises. Further, the presented architecture can be easily deployed on alternative SPARQL endpoints by only changing the queries (which retrieve temporal data) as long as their results are structured similarly.

We presented in this article the first version of our interface focused on representing Wikidat's temporal information using timelines. Practically, we are cur-

[6] Project's code base: https://github.com/wikitimeline/wikitimeline.github.io

rently setting up a user validation experiment in order to improve the timeline *navigation* and experience. On a different note, we are also planning to improve the WebApp-side with additional features such as: allowing timeline exports as image or improving the coloring to for instance group predicate together. Finally, it is worth noticing that our architecture is not bound to querying a single endpoint at once; indeed, it is possible to extend the *compare* feature presented above to query two or more SPARQL endpoints, allowing a comparison of the temporal information available in different databases at a glance.

Acknowledgments This research was conducted with the financial support of the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 713567 at the ADAPT SFI Research Centre at Trinity College Dublin. The ADAPT SFI Centre is funded by Science Foundation Ireland through the SFI Research Centres Programme and co-funded under the European Regional Development Fund Grant #13/RC/2106.

References

1. Bikakis, N., Skourla, M., Papastefanatos, G.: rdf:synopsviz – A framework for hierarchical linked data visual exploration and analysis. In: European Semantic Web Conference. pp. 292–297. Springer (2014)
2. Gottschalk, S., Demidova, E.: EventKG: A multilingual event-centric temporal knowledge graph. In: ESWC. pp. 272–287. Springer (2018)
3. Gottschalk, S., Demidova, E.: EventKG–the hub of event knowledge on the web–and biographical timeline generation. Semantic Web **10**(6), 1039–1070 (2019)
4. Haase, P., Herzig, D.M., Kozlov, A., Nikolov, A., Trame, J.: metaphactory: A platform for knowledge graph management. Semantic Web **10**(6), 1109–1125 (2019)
5. Hartig, O.: Foundations of RDF* and SPARQL*:(An alternative approach to statement-level metadata in RDF). In: AMW 2017 11th Int. Workshop on Foundations of Data Management and the Web. vol. 1912 (2017)
6. Junior, A.C., Orlandi, F., Graux, D., Hossari, M., O'Sullivan, D., Hartz, C., Dirschl, C.: Knowledge graph-based legal search over german court cases. In: ESWC (2020)
7. Kubitza, D.O., Böckmann, M., Graux, D.: Semangit: A linked dataset from git. In: International Semantic Web Conference. pp. 215–228. Springer (2019)
8. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web Journal **6**(2), 167–195 (2015)
9. Manola, F., Miller, E., McBride, B., et al.: RDF primer. W3C recommendation **10**(1-107), 6 (2004)
10. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A core of semantic knowledge. In: WWW. pp. 697–706. ACM, New York, NY, USA (2007)
11. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)
12. W3C SPARQL Working Group, et al.: SPARQL 1.1 overview (2013), http://www.w3.org/TR/sparql11-overview/
13. Wishart, D.S., Knox, C., Guo, A.C., Cheng, D., Shrivastava, S., Tzur, D., Gautam, B., Hassanali, M.: Drugbank: a knowledgebase for drugs, drug actions and drug targets. Nucleic acids research **36**(suppl_1), D901–D906 (2008)

VOWLMap: Graph-based Ontology Alignment Visualization and Editing *

Ana Guerreiro[1], Daniel Faria[1], and Catia Pesquita[1]

LASIGE, Dep. de Informática, Faculdade de Ciências da Universidade de Lisboa, Portugal

Abstract. Manual validation of automated ontology alignments remains essential to ensure their quality. However, few alignment systems feature user interfaces enabling alignment visualization, validation and editing, and those that do, support a limited number of requirements.
We developed VOWLMap—an extension for the standalone web application, WebVOWL—for visualizing, editing, and validating ontology alignments. WebVOWL implements the Visual Notation for OWL Ontologies (VOWL) which defines a visual representation for most language constructs of OWL. We extended VOWL to support graphical representations of alignments and restructured WebVOWL to load and visualize alignments. VOWLMap employs modularization techniques to facilitate the visualization of large alignments while maintaining the context of each individual mapping, and supports diverse interaction mechanisms, including direct interaction with and editing of graph representations.
We conducted a user study to collect feedback on VOWLMap, using as tasks the validation of alignments from the biomedical and conference domains.

Keywords: Ontology Alignment · Ontology Matching · Alignment Visualization

1 Introduction

Ontology alignment (or matching) is a solution to the semantic heterogeneity problem, as it establishes relations between entities of related ontologies, enabling interoperability [14]. This is critical due to the growing traction of ontologies and knowledge graphs as solutions for information management, which has led to their widespread but uncoordinated development.

Several ontology alignment algorithms and systems have been proposed over the last two decades, but for the most part, alignment systems focus on automated approaches without human intervention [4, 14]. Due to the complexity of ontologies, automated alignments often contain erroneous mappings, and are seldom complete [12, 14]. User validation of ontology alignments is essential to overcome the limitations of automated algorithms, as users can remove or correct erroneous mappings, as well as add missing ones [7].

Given the size and complexity of ontologies and alignments, it has become clear that comprehensive and more interactive visualizations are key features for user involvement

in alignment validation, as they enable a better understanding of the alignment and support the decision-making process. Nevertheless, few alignment systems provide a user interface that supports alignment visualization, editing and navigation strategies, and even fewer provide the functionalities needed to make the task seamless for the user, such as interaction with the visualization or contextual information about the mappings [7].

According to [7], there are two aspects regarding user interfaces that are determinant to the process of alignment validation: alignment visualization and alignment interaction. Our main goal was to develop an interactive tool with an interface that provides visual support and functionalities to allow users to interact with and validate an alignment.

In this paper, we present VOWLMap[1], a tool for visualizing, validating and editing ontology alignments. VOWLMap extends the web application, WebVOWL [9] and its underlying visual notation (VOWL) [8] to the context of ontology alignment, offering an intuitive and comprehensive visualization that can be understood by users less familiar with ontology alignments. VOWLMap employs modularization techniques to facilitate the visualization of large alignments while maintaining the context of each individual mapping, and supports diverse interaction mechanisms, including direct interaction with and editing of graph representations.

2 Related Work

Ontology alignment visualizations can usually be divided into two visual paradigms: trees and graphs [4].

Under the first paradigm, ontologies are usually displayed side by side as indented trees—providing an intuitive representation of the hierarchy—and mappings are typically represented as lines connecting the corresponding nodes. Several ontology matching systems implement tree visualizations, including AgreementMaker [2] and COMA 3.0 Community Edition [10].

Under the second paradigm, systems that implement graph visualizations typically offer two views of an alignment: a list view spanning the whole alignment, and a graph view corresponding to a sub-graph of the alignment and ontologies. Examples include AgreementMakerLight [3], whose graph representation captures the neighborhood of a mapping, and YAM++ [1], that provides independent graph visualizations for the entities in a mapping.

Orthogonally, Ivanova et al. have focused on the visual exploration and evaluation of multiple ontology alignments. They proposed an interactive visualization interface to simultaneous explore and evaluate multiple alignments at different levels of granularity [5].

Visualizing ontology alignments requires first and foremost visualizing ontologies, since they usually dwarf alignments in volume of information and complexity. Moreover, two ontologies plus their alignment can be seen essentially as a larger ontology.

[1] Available at https://github.com/liseda-lab/VOWLMap

The Visual Notation for OWL Ontologies (VOWL) [8] is a visual language for user-oriented representation of ontologies that aims to help users intuitively understand ontology semantics. VOWL defines a set of graphical primitives and a color scheme that express specific attributes for most language constructs of OWL.

WebVOWL [9] implements VOWL as a standalone web application for interactive visualization of ontologies. It defines a JSON schema into which OWL ontologies need to be converted, which can be performed using OWL2VOWL, a Java-based converter deployed alongside WebVOWL. WebVOWL renders the graphical elements according to the VOWL specifications in a dynamic force-directed graph layout, using the JavaScript library D3. It implements basic interaction techniques such as zoom, pan and drag and drop. A sidebar displays details on a selected entity on the graph, along with other information, such as metadata, description, and metrics. Users can filter the visualization to reduce the size of the graph, based on property types or node degree. WebVOWL also supports text-based search and selecting a matching entity locates it and highlights it in the graph. In the experimental editing mode it is possible to create, edit and delete elements. Finally, the visualization can be saved as an SVG image or the ontology can be exported in JSON format.

3 VOWLMap

3.1 Methodology

Our approach to develop VOWLMap began with an analysis of the requirements described in [6, 7] and a selection of target requirements, followed by an assessment of existing development and visualization options. The success of Javascript-based visualization for complex data afforded by d3.js led us in the direction of a browser-based architecture, which facilitates use by precluding the need for software installation. After investigating existing browser-based ontology visualization systems, we reached the conclusion that WebVOWL matched our functional and technical requirements.

Regarding the categories of issues that affect alignment validation [7], we opted to target use cases where: (1) users are domain experts but may be ontology alignment novices; and (2) users are only involved after the alignment process, interacting with the alignment but not with the alignment system. We identified the following functional requirements:

(R1) Loading of ontologies and corresponding alignment;
(R2) Provision of alternative alignment views to support different tasks and user preferences;
(R3) Support for visual information seeking tasks [13], i.e., (R3.1) overview (overview of the entire collection), (R3.2) zoom (zoom in or out on items of interest), (R3.3) filter (remove uninteresting elements from the visualization), (R3.4) details-on-demand (select element and obtain details), (R3.5) relate (view relationships among elements), (R3.6) history (keep track of actions) and (R3.7) extract (extraction of sub-sets of elements).
(R4) Indication of mapping status, that is distinguishing between validated and candidate mappings;

(R5) Visualization of metadata, such as definitions and synonyms;
(R6) Visualization of a mapping context, i.e., showing the neighbourhood of the entities involved in the mapping, including nearby mappings;
(R7) Accepting and rejecting mappings;
(R8) Creating and refining mappings, i.e., adding new mappings manually or refining an existing mapping by altering the source or target entity.
(R9) Search, that is the ability to search for ontology entities by their labels;
(R10) Session support, that is accommodating interruptions when validating;
(R11) Exporting into different alignment formats.

3.2 VOWL Extension

Since the VOWL notation was conceived to represent a single ontology, we had to extend it to (1) distinguish between the two aligned ontologies (by representing them in different colors), and (2) represent mappings between them, with different colors for mappings with different (revision) status. We assigned the general light blue to the source ontology, and the dark blue to the target ontology (Figure 2). Each mapping is represented by a solid line linking the two mapped nodes, with arrowheads at both ends and a rectangle indicating the confidence score. To color these elements according to the mapping status, we add four new colors: dark green for *correct*, dark red for *incorrect*, medium yellow for *unsure* and medium gray for *unreviewed*.

The color scheme of VOWL already includes some variations of the colors mentioned above, which could potentially lead to misinterpretations between: (1) *incorrect* mappings in dark red and highlighting in VOWL red; (2) *unsure* mappings in medium yellow and datatype in VOWL yellow; (3) *unreviewed* mappings in medium gray and deprecated elements in VOWL light gray; and to a lesser extent (4) *correct* mappings in dark green and data properties in VOWL light green. However, we believe that the fact that mappings include a boxed label with the confidence score should enable a clear distinction between mappings and all these cases. Note also than under this extension, external elements to the ontology, represented by dark blue in VOWL, no longer have a representation in either the source or the target ontology.

3.3 Functionalities

WebVOWL supports some of the functional requirements we identified, however it does not support those directly related to alignment visualization and editing. VOWLMap extends WebVOWL with several features to support these processes.

Like WebVOWL, VOWLMap requires a JSON representation of the ontologies and alignment as input. To enable this, we developed a small Python-based tool that receives as input the JSON files of the two ontologies (previously converted using OWL2VOWL [9]) and an alignment RDF file. This tool merges these files into a single JSON file that includes all the information about the ontologies and the alignment, and that can be loaded by VOWLMap. To tackle the challenge of loading large ontologies, this tool only loads elements of the ontologies into the JSON file that are at a maximum distance of 3 edges of each mapping. This facilitates memory management and optimizes the

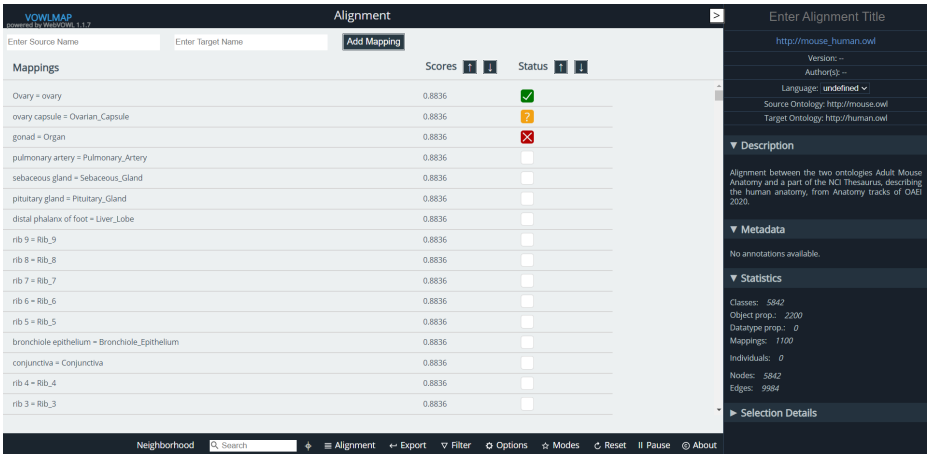


Fig. 1: VOWLMap alignment panel.

uploading time, by removing less relevant information, since the graph visualization only extends the neighborhood to a maximum of 3 edges.

VOWLMap provides two views of alignments: an alignment panel (Figure 1), and a graph visualization (Figure 2). Both views include a sidebar, listing information about the alignment, and a footer menu containing the visualization controls. The alignment panel is composed by a list of mappings, with their confidence score and status. In this panel, users can validate and create new mappings (Figure 1). By clicking a mapping from the list, a graph visualization for that mapping is generated, where both ontologies are represented in the same graph with different colors, and mappings are represented as double-edges arrows colored according to their status and labeled with their score. In addition to the selected mapping, all other mappings present in its neighborhood are shown.

Users can interact with the visualization: zoom in and out, pan the background and move elements around to adapt the force-directed layout. It is possible to change the characteristics of the visualization, such as class distance or datatype distance, and adjust the neighborhood (from zero to a maximum of three edges of distance). Moreover, VOWLMap provides the same filters as WebVOWL, that can be activated or deactivated at any time if users want to exclude or include information in the visualization or focus on certain aspects. In addition to searching for ontology entities, VOWLMap allows searching for a specific mapping in both views. When users enter the name of the one of the entities participating in the mapping in the search bar, a graph for that mapping is generated.

Users can visualize and edit a mapping status in both views. VOWLMap supports 4 values for the status - *unreviewed*, *correct*, *incorrect* and *unsure*. When users select a mapping in the graph, the sidebar provides information about the mapping, including a dropdown with its status where users can validate the mapping by selecting one of the four available options. For instance, Figure 2 shows the status of the mapping *Ovary - ovary*, that was set with the value *correct*. In the alignment panel, users can also validate

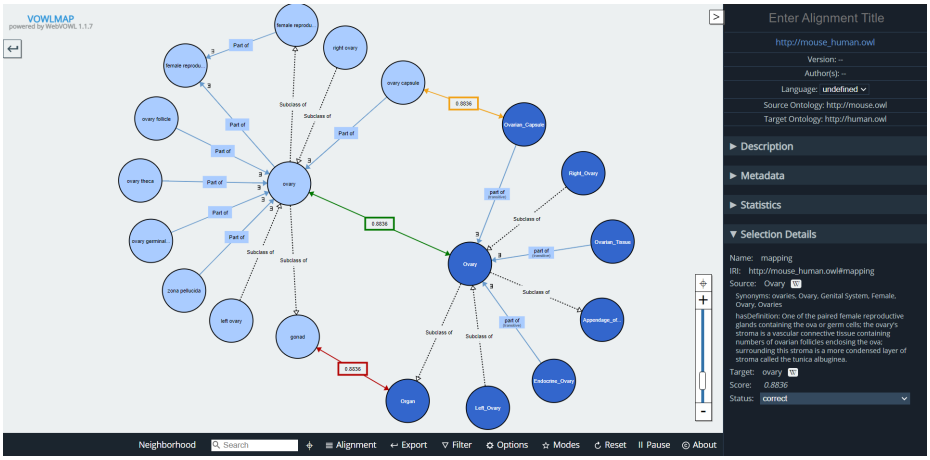


Fig. 2: VOWLMap visualization of a mapping.

a mapping in the sidebar, or in the status checkbox, by clicking on it. Each status has an associated color and icon to distinguish them.

The sidebar provides information about the source and target classes of a selected mapping, such as synonyms and definitions. For each mapped class, VOWLMap generates an automatic link to Wikipedia, which can help users obtaining more information. Figure 2 shows a clickable icon next to each class to open a Wikipedia page for that term in a new tab, if available. Furthermore, users can visualize an individual mapping and its local context, including neighboring mappings (Figure 2). By default, VOWLMap displays the neighborhood at a distance of 1, but users can change it from 0 to a maximum of 3 edges.

VOWLMap allows users to manually remove or add new mappings directly in the visualization. When a new mapping is created, the maximum score and the status *correct* are assigned to that mapping. In the alignment panel, users can create new mappings by entering the label of the respective source and target classes, and VOWLMap automatically generates a graph for that mapping. Moreover, users can refine an existing mapping by dragging the ends of its source or target nodes to a more suitable one.

The changes made to the alignment are saved in a cached version and, as long as VOWLMap is open in the browser, this version is always loaded, supporting interruptions in the validation process. A reload button allows users to discard these new changes and reload the original alignment. Moreover, VOWLMap allows exporting the validated alignment in several formats (e.g. RDF, JSON) or additionally, the complete or filtered rendering of a mapping in SVG.

Table 1 compares how VOWLMap and state-of-the-art ontology alignment systems comply with the requirements for alignment validation detailed in [7]. VOWLMap is the only tool that complies (totally or partially) with all requirements.

Table 1: Addressing of alignment validation requirements by VOWLMap and state-of-the-art ontology alignment systems.

	AM	AML	YAM++	COMA	VOWLMap
(R1) Load Alignments	✓	✓	✓	✓	✓
(R2) Alternative Views	×	✓	×	×	✓
(R3) Visual Info-Seeking Tasks	✓	✓*	✓*	✓*	✓*
(R4) Mapping Status	✓	✓	✓	×	✓
(R5) Metadata	✓	✓	×	×	✓
(R6) Context	✓*	✓	×	×	✓
(R7) Accept/Reject mapping	✓	✓	✓	✓*	✓
(R8) Create/Refine mapping	✓*	✓*	✓*	✓	✓
(R9) Search	×	✓	✓	✓	✓
(R10) Session	✓*	✓*	✓*	✓	✓
(R11) Export Alignments	✓	✓	✓	✓	✓

4 Evaluation

We performed a formative assessment study to guide the further development of VOWLMap. This study was observational and task-oriented, falling within the scope of the *creation* and *management* tasks of user studies in a Semantic Web context [11], given that the purpose of VOWLMap is to visualize, validate and edit ontology alignments.

4.1 Methodology

Four users were recruited from a pool of graduate students, with different backgrounds (life sciences, health sciences, computer science and engineering) and levels of expertise in alignment validation. All participants had prior knowledge of at least one of the domains of the aligned ontologies.

The evaluation focused on two small alignment validation tasks derived from the Conference and Anatomy tracks of OAEI 2020[2]. The first focused on an alignment between the ontologies *Conference* and *ekaw*, whereas the second focused on an alignment between the Adult Mouse Anatomy ontology and a part of the NCI Thesaurus describing human anatomy. Both alignments contained precisely 20 mappings, 10 of which were correct and selected from the reference alignment, with the other 10 being incorrect and selected from the erroneous mappings found by AML [3]. The selection of mappings was manual, and sought to ensure that there were both trivial (same label)

[2] http://oaei.ontologymatching.org/2020/

and non-trivial mappings among both correct and incorrect mappings, and that all incorrect mappings reflect some of the errors that automated tools make. The users were not aware of the proportion of positive and negative mappings in the tasks.

The study was performed remotely, with users employing VOWLMap on their machines. Participants were monitored and assisted by a researcher from our team via the *Zoom* platform, with participants sharing their screen but with cameras disabled. Audio and video were recorded with informed consent from the participants, collected through an online questionnaire[3]. This questionnaire also served to assess the profile of the user and collect details about the hardware, computer screen and web browser used by the participants, as well as to deliver the instructions of the tasks to the users and collect their feedback. The instructions required the users to watch a tutorial video[4], then download and run VOWLMap locally, validating the alignments provided for the study.

No instructions were given regarding the type or extent of validation required. Any questions regarding VOWLMap were answered. After each task, users uploaded the RDF file of their final alignment. Finally, users rated the features of VOWLMAP in a Likert scale, ranging from "Not useful" to "Very useful". At the end of the questionnaire, an open-ended question allowed users to provide suggestions or feedback about VOWLMap.

4.2 Results and Discussion

To analyze the evaluation, we computed the duration of each task and each mapping validation action, the revision made by each user (i.e. count of mappings classified as correct, incorrect, and unsure, as well as new mappings added and existing mappings refined), the correctness of each validation action assessed with reference alignment (true and false positives and negatives, plus correct new and refined mappings), and the frequency of use of each VOWLMap feature. These results are compiled in Table 2 and the timeline of each validation task is depicted in Figure 3, presenting the time spent on each mapping and time spent in interruptions for requesting help or clarifications.

Four users were selected for this study, with different backgrounds (life sciences, health sciences, computer science and engineering, and bioinformatics) and levels of expertise in alignment validation. User 3 was the most experienced in ontology alignments, followed by User 4, whereas Users 1 and 2 had no previous experience. This is reflected in the outcomes of the validation, as Users 3 and 4 were quicker in the validation of both alignments, and User 3 was the only one to refine mappings, having 100% accuracy in mappings refined or added.

Table 3 displays the ratings given by each user to VOWLMap's several features, and Figure 4 depicts the frequency of use of each of those features per user and task.

All users had a fairly high accuracy classifying the mappings, ranging from 80% to 95% across the two tasks. The average accuracy was greater in Anatomy (91.3%) than Conference (86.3%) which mirrors the fact that automated alignment systems have worse results in the latter, suggesting it is a more difficult task. Users did take more

[3] https://bit.ly/36HreUP

[4] Available at: https://youtu.be/aCFtHtuN5Gk

Table 2: Evaluation statistics per task and user: mappings classified as Correct (Cor), Incorrect (Inc) or Unsure (Uns); New mappings added; mappings Refined (Ref); False Positives (FP), False Negatives (FN), True Positives (TP) and True Negatives (TN); New and Refined mappings that are Correct (New Cor and Ref Cor); and duration of the task (Time).

		Cor	Inc	Uns	New	Ref	FP	FN	TP	TN	New Cor	Ref Cor	Time (mm:ss)
Task 1 Conference	User 1	7	12	1	6	0	0	3	7	9	1	-	45:54
	User 2	11	9	0	1	0	1	0	10	9	1	-	30:50
	User 3	9	9	2	0	0	1	0	8	9	-	-	14:37
	User 4	9	10	1	0	0	1	1	8	9	-	-	10:32
Task 2 Anatomy	User 1	12	8	0	13	0	2	0	10	8	8	-	23:51
	User 2	11	9	0	2	0	1	0	10	9	2	-	43:53
	User 3	10	10	1	2	3	1	0	10	8	2	3	19:30
	User 4	9	9	2	0	0	0	0	9	9	-	-	08:35

Table 3: Rating of VOWLMap's features by each user.

	List Vis.	List Valid.	List Editing	Graph Vis.	Graph Valid.	Graph Editing	Graph Interaction	Wikipedia Links
User 1	5	5	5	5	5	5	5	4
User 2	4	4	5	5	5	5	5	4
User 3	5	4	5	5	5	5	5	4
User 4	5	5	4	5	4	3	5	4
Mean	4.73	4.72	4.73	5	4.73	4.40	5	4

time in average in Anatomy (122 minutes) than Conference (112 minutes), but this is likely due to a greater number of new and refined maps contributed (20 for Anatomy versus only 7 for Conference) which are time consuming. Interestingly, users identified erroneous mappings better in Conference than Anatomy, whereas the reverse was true for correct mappings.

User 2 was the only one not to classify any mapping as unsure, and also the one with the highest overall accuracy, with 95% in both tasks. Regarding the creation or refinement of mappings, User 1 was the most prolific but had an overall precision of only 47.4%. Users 2 and 3 created or refined only a few mappings, but with 100% precision, whereas User 4 neither created nor refined mappings. As users were asked to not remove the original mappings but rather mark them as incorrect, the use of the removing function is not reflected by Table 2. However, this feature was used by 3 users to remove mappings they previously added.

All users requested help or clarifications about VOWLMap, but time spent on this was comparatively short (2.5% - 9.8%).

Familiarity with VOWLMap seemingly had no bearing on the speed or accuracy with which users classified mappings, as initial mappings had neither higher time nor

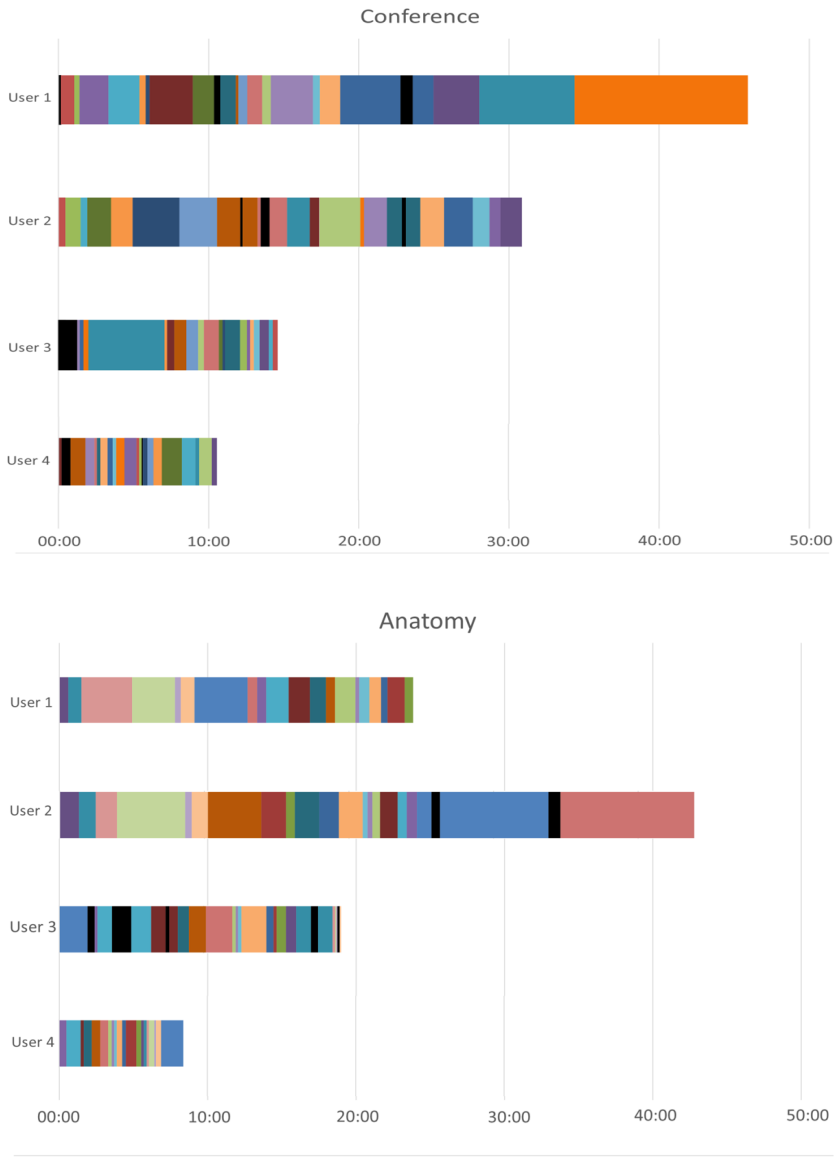


Fig. 3: Time spent validating Task 1 (Conference) and Task 2 (Anatomy), by each user. Each color represents a different mapping, with black representing interruptions for requesting help or clarifications about VOWLMap. The mapping corresponding to each color is at https://bit.ly/36HreUP

lower accuracy on average. On the contrary, Users 3 and 4 seem to have left mappings they found more challenging to classify for last, taking a lot more time in these. In-

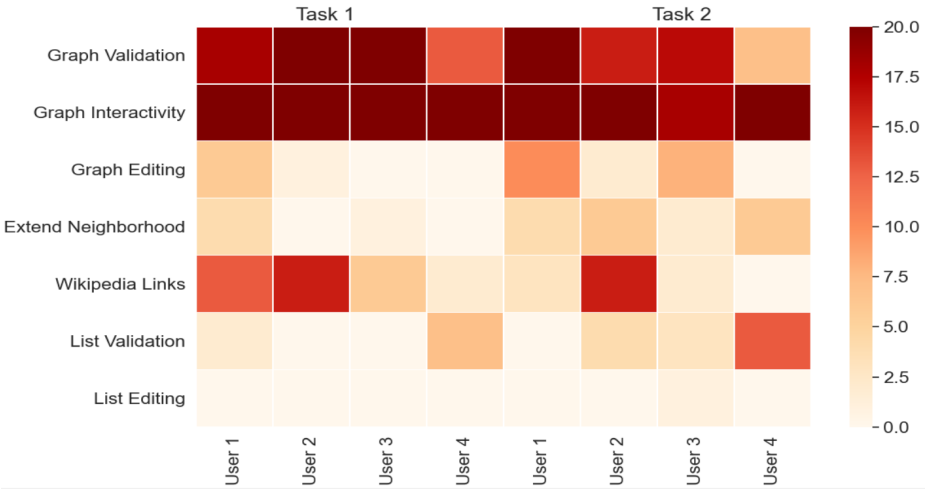


Fig. 4: Frequency of use of each VOWLMap feature, measured in number of mappings where the feature was used, for Task 1 (Conference) and Task 2 (Anatomy).

terruptions for help or clarification were also not concentrated at the start of the task, although Users 1 and 2 did ask for clarifications at the start of the evaluation. Overall, these facts speak well to the intuitiveness of VOWLMap's visualizations and functionalities.

With respect to VOWLMap's features, we note that *Wikipedia Links* received the lowest rating from users (4) likely because the links are automatically generated from the entities' labels, and sometimes there is no Wikipedia page available for a term. The only feature that was scored 3 by any user was *Graph Editing*, by user 4, who notably was the only user that attempted neither additions nor refinements of mappings, making the least use of this functionality among the four users. This functionality was more used in Anatomy than in Conference by the other three users, as they created and refined more mappings in the former than the latter. *Graph Visualization* and *Graph Interactivity* were consistently the highest rated features and the most used by users in both tasks.

5 Conclusions

We developed VOWLMap, a browser-based tool for ontology alignment visualization, validation and editing. VOWLMap provides both list and graph visualization of mappings, and supports the annotation of mappings as correct, incorrect and unsure, as well as the creation or refinement of mappings. VOWLMap complies with all the requirements for user validation of ontology alignments, laid out in [7] (albeit partially, in the case of the visual information seeking tasks), which sets it above established ontology alignment systems.

VOWLMap's evaluation in a small user study revealed that it is intuitive and easy to use, as no learning curve was observed with respect to the time or accuracy of the validation tasks. Moreover, users made use of most of VOWLMap's features, and generally considered them useful in the feedback they provided.

Future work could include adding features to provide overall statistics to assist in the validation process monitoring, such as displaying the mapping coverage for the aligned ontologies, the number of mappings reviewed by the current or previous user, and the number of changes made to that point. Additionally, it would be interesting to support validation of inter-annotator agreement, by allowing experts to exchange they intermediate results or by storing the revision along with the changes made by each author. Furthermore, we plan to integrate some of the qualitative assessments made by the users to further improve VOWLMap, as well as design a broader usability study, with in-person observations, to delve deeper into how users interact with the tool.

Acknowledgments This work was supported by FCT through the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020). It was also partially supported by the KATY project which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 101017453.

References

1. Bellahsene, Z., Emonet, V., Ngo, D., Todorov, K.: Yam++ online: A web platform for ontology and thesaurus matching and mapping validation. In: Blomqvist, E., Hose, K., Paulheim, H., Ławrynowicz, A., Ciravegna, F., Hartig, O. (eds.) The Semantic Web: ESWC 2017 Satellite Events. pp. 137–142. Springer International Publishing, Cham (2017)
2. Cruz, I.F., Antonelli, F.P., Stroe, C.: Agreementmaker: Efficient matching for large real-world schemas and ontologies. Proc. VLDB Endow. **2**(2), 1586—1589 (2009)
3. Faria, D., Pesquita, C., Santos, E., Palmonari, M., Cruz, I.F., Couto, F.M.: The agreementmakerlight ontology matching system. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences. pp. 527–541. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
4. Granitzer, M., Sabol, V., Onn, K.W., Lukose, D., Tochtermann, K.: Ontology alignment—a survey with focus on visually supported semi-automatic techniques. Future Internet **2**(3), 238–258 (2010)
5. Ivanova, V., Bach, B., Pietriga, E., Lambrix, P.: Alignment cubes: Towards interactive visual exploration and evaluation of multiple ontology alignments. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) The Semantic Web – ISWC 2017. pp. 400–417. Springer International Publishing, Cham (2017)
6. Ivanova, V., Lambrix, P., Åberg, J.: Requirements for and evaluation of user support for large-scale ontology alignment. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) The Semantic Web. Latest Advances and New Domains. pp. 3–20. Springer International Publishing, Cham (2015)
7. Li, H., Dragisic, Z., Faria, D., Ivanova, V., Lambrix, P., Jiménez-Ruiz, E., Pesquita, C.: User validation in ontology alignment: functional assessment and impact. The Knowledge Engineering Review **34**, e15 (2019)

8. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Vowl 2: User-oriented visualization of ontologies. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) Knowledge Engineering and Knowledge Management. vol. 8876, pp. 266–281. Springer International Publishing, Cham (2014)
9. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with vowl. Semantic Web **7**(4), 399–419 (2016)
10. Massmann, S., Raunich, S., Aumüller, D., Arnold, P., Rahm, E.: Evolution of the coma match system. In: Proceedings of the 6th International Conference on Ontology Matching. OM'11, vol. 814, p. 49–60. CEUR-WS.org, Aachen, DEU (2011)
11. Pesquita, C., Ivanova, V., Lohmann, S., Lambrix, P.: A framework to conduct and report on empirical user studies in semantic web contexts. In: Faron Zucker, C., Ghidini, C., Napoli, A., Toussaint, Y. (eds.) Knowledge Engineering and Knowledge Management. pp. 567–583. Springer International Publishing, Cham (2018)
12. Pour, N., Algergawy, A., Amini, R., Faria, D., Fundulaki, I., Harrow, I., Hertling, S., Jimenez-Ruiz, E., Jonquet, C., Karam, N., et al.: Results of the ontology alignment evaluation initiative 2020. In: Proceedings of the 15th International Workshop on Ontology Matching (OM 2020). vol. 2788, pp. 92–138. CEUR-WS (2020)
13. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. IEEE Symposium on Visual Languages, Proceedings (2000)
14. Shvaiko, P., Euzenat, J.: Ontology matching: State of the art and future challenges. Knowledge and Data Engineering, IEEE Transactions on **25**, 158–176 (2013)

VizKG: A Framework for Visualizing SPARQL Query Results over Knowledge Graphs

Hana Raissya[1], Fariz Darari (✉)[1,2], and Fajar J. Ekaputra[3]

[1] Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia
[2] Tokopedia-UI AI Center of Excellence, Jakarta, Indonesia
{hana.raissya, fariz}@ui.ac.id
[3] Institute of Information Systems Engineering, TU Wien, Vienna, Austria
fajar.ekaputra@tuwien.ac.at

Abstract. Despite the rise of the knowledge graph (KG) popularity, understanding SPARQL query results from a KG can be challenging for users. The use of data visualization tools, e.g., Wikidata Query Service and YASGUI, can help address this challenge. However, existing tools are either focused just on a specific KG or only provided as a web interface. This paper proposes VizKG, a framework that provides a wide range of visualizations for SPARQL query results over KGs. VizKG aims to assist users in extracting patterns and insights from data in KGs, and hence supporting further KG analysis. VizKG features a wrapper that links SPARQL query results and external visualization libraries by mapping query result variables to the required visualization components, currently allowing for 24 types of visualizations. Not only that, VizKG also includes visualization recommendations for arbitrary SPARQL query results as well as extension mechanisms for additional visualization types. In our evaluation, the visualization recommendation feature of VizKG achieves an accuracy of 87.8%. To demonstrate the usefulness of VizKG in practical settings, this paper also reports on use case evaluation over various domains and KGs. A Python-based, Jupyter Notebook friendly implementation of VizKG is openly available at https://pypi.org/project/VizKG/.

Keywords: Visualization · Knowledge Graphs · SPARQL · Insights

1 Introduction

A knowledge graph (KG) mainly describes real-world entities and their interrelations in a graph structure, allowing to cover various domains [3]. The Semantic Web and Linked Data are concrete embodiments of KGs, popularized by Google in 2012 through the Google Knowledge Graph.[4] In the field of data science, the development of the Semantic Web and Linked Data is becoming increasingly important, serving as both primary and contextual data sources [13].

[4] https://blog.google/products/search/introducing-knowledge-graph-things-not/

Visualization is one of the stages in the data science pipeline that plays a key role in data exploration and analysis [4]. Data visualization can help mediate between data scientists and domain experts pertaining to the validation of assumptions and findings. In particular, data visualization might come into handy when little is known about data sources and analytical objectives [5].

Generally, visualizing (tabular) data takes form of charts, such as time-series charts, geographic maps, statistical charts, as well as hierarchies and networks [6]. Visualizations can also be performed over KGs, e.g., Wikidata. Wikidata serves as a (semantic) data hub among all editions of Wikipedia as well as external sources [8]. Wikidata features Wikidata Query Service (WQS), which not only supports SPARQL queries, but also visualizations in the form of image grids, timelines, dimensions, treemaps, and many more. WQS is indeed helpful to provide graphical information about the data stored in Wikidata. Nevertheless, its usage is limited to Wikidata only. On the other hand, visualization services such as LODmilla[5] and YASGUI[6] facilitate visualizing generic KGs though with a limited support of visualization types compared to WQS.

The provision of KG visualization services using a web, standalone tool is however less ideal for the data science community. In terms of data visualization, the Jupyter Notebook[7] is a popular choice and has advantages over other platforms [9], such as real-time interaction with code, inline printing of output, and PDF or HTML exports. Visualizations generated from a Jupyter notebook can be presented in a browser or as a shared Google Colaboratory.[8] With these advantages, Jupyter notebooks enable users to narrate visualizations generated from (Python) libraries, e.g., Matplotlib[9] and Plotly.[10]

Based on the need for exploration and visualization on the increasingly popular KGs, through this study, we propose VizKG (https://pypi.org/project/VizKG/), a Python-based framework with the following functionalities: (i) visualization of SPARQL query results on generic KGs; (ii) automatic recommendation of visualization types; and (iii) extension mechanisms for additional visualization types.

2 Related Work

KG visualization is a research topic that has been around for many years. A stream of research studies aims to support end-users to explore and visualize KGs. One of the earlier studies in this area is SemLens [7], which provides a visual tool allowing end-users to perform robust data analysis on KGs. SemLens, however, focuses only on a single visualization type, i.e., scatter chart. Linked Data Visualization Model (LVDM) aims to provide a formal model for RDF data visualization [2]. LVDM provides two reference implementations: (i) LODVisualization, connecting & analyzing various datasets, and (ii) Payola, providing details on specific parts of KGs. Another approach is LDVizWiz [1],

[5] https://www.dbpedia.org/community/lodmilla/
[6] https://yasgui.triply.cc/
[7] https://jupyter.org/
[8] https://colab.research.google.com/
[9] https://matplotlib.org/
[10] https://plotly.com/

which identifies seven data categories and their associated standard vocabularies for visualizations in existing Linked Data infrastructure and workflow. LinkDaViz [14] is a tool that allows for automatic data visualization. The tool features a recommendation algorithm that automatically binds data properties to visualization options. Nevertheless, the tool does not support direct visualization of arbitrary SPARQL query results. More recently, ProWD [11] is a user-friendly tool developed to visualize knowledge imbalances in Wikidata. The tool supports visualization types like bar charts & area charts, and demonstrates how KG visualizations can be relevant in practice. However, the tool only caters to very specific use cases (i.e., knowledge imbalances) for a specific KG (i.e., Wikidata).

Another stream of research focuses on supporting KG visualizations for more technical, SPARQL-savvy users. In this direction, the most prominent approaches are web-based approaches, such as YASGUI [12] and Wikidata Query Service (WQS). YASGUI is a web-based SPARQL client that can be used to query both remote and local endpoints. It allows visualizing SPARQL query results, albeit with limited visualization types. Unlike a general-purpose SPARQL tool such as YASGUI, the WQS interface has been customized for Wikidata to improve its functionalities. WQS supports a wide variety of result visualizations in addition to the standard tabular view. The WQS interface, however, can only be used on Wikidata. While these tools are helpful for technical users, there are still gaps to support (generic) KG visualizations by the growing role of data scientists, who typically rely on a specific environment (e.g., Jupyter Notebook) for their data science pipeline.

3 VizKG Framework

In this section, we present the general VizKG architecture and workflow (Section 3.1), VizKG visualization recommendation procedure (Section 3.2), and the extension mechanism for new visualization types (Section 3.3).

3.1 Architecture and Workflow

Fig. 1 displays the architecture as well as workflow of VizKG. VizKG consists of four main stages: (i) Preprocessing, (ii) Query Execution, (iii) Visualization Recommendation, and (iv) Visualization Generation.

From the user input, VizKG receives a SPARQL query string, a SPARQL endpoint URL, and (optionally) a preferred chart type (Step 1). Next, the *Preprocessing* stage of VizKG parses the query string and validates the endpoint URL (Step 2). VizKG also checks whether the selected chart type is supported or not. Then, in the *Query Execution*, VizKG invokes a REST-based API call to a remote SPARQL endpoint, to which the SPARQL query is evaluated, and that subsequently the query results (in JSON format) are returned (Step 3). Afterwards, VizKG transforms the query results into a tabular form. VizKG leverages the SPARQLWrapper[11] library to support both the Preprocessing and Query Execution stages.

[11] https://github.com/RDFLib/sparqlwrapper

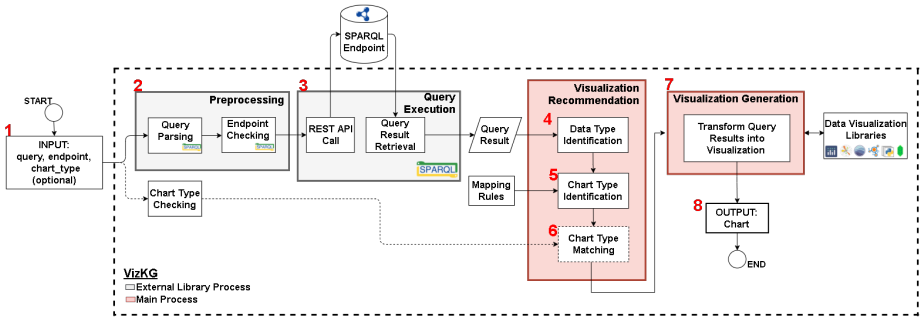


Fig. 1. VizKG Architecture (bit.ly/VizKG-Architecture)

The next stage is the *Visualization Recommendation*. The first thing to do in this stage is to identify the datatypes of each query variable (Step 4). The identified datatypes as well as ordering of the variables serve as the main reference to determine recommended charts (or visualizations) of the query results based on the VizKG mapping rules (Step 5). In the case that a preferred chart type is given as an input parameter, VizKG simply checks if the chart type is in the chart recommendations (Step 6). As for the other case when there is no chart type preference given, VizKG bypasses the chart type matching. In the *Visualization Generation* stage, VizKG maps the query result variables to visualization components, and delegates the creation of the visualization to off-the-shelf visualization libraries, such as Matplotlib, Plotly, and Seaborn (Step 7). The final output would be the graphical visualization of the query results (Step 8).

3.2 Visualization Recommendation

The *Visualization Recommendation* stage determines which visualization types are suitable for the returned query results. Here we describe the stage in more detail.

First, the datatypes wrt. the variable values are identified via regex matching. We support the following datatypes: numerical, date, URI, image, coordinate, and label. Then, via the VizKG mapping rules,[12] we check the compatibility of the datatypes of the query variables with the datatypes of the supported charts.

In the checking process, we use the following heuristics: (i) all visualization components (except the optional ones) must have query variables mapped into them; (ii) the datatype of the query variable mapped must conform to the required datatype of the chart; and (iii) whenever there are more than one conforming datatypes, the ordering matters (i.e., it is first-come-first-map).

As an example, if there is a query result with two variables (i.e., V_1 and V_2) of type numerical, then the visualization of scatter chart can be recommended, since there are exactly two non-optional variables required with the same datatypes, and that the variable V_1 is mapped to the X-component and V_2 to the Y-component of the scatter chart. VizKG iterates over all visualization types, checking whether they are compatible or not, and collects the compatible ones as visualization recommendations.

[12] https://bit.ly/VizKG-MappingRules

3.3 Extension Mechanism for New Visualization Types

In the code structure of VizKG,[13] all chart implementations are made modular, in that they have their own classes, inheriting from the `Chart` class. Every class added to VizKG must then make concrete the methods of the datatype compatibility checking, variable-to-component mapping, and visualization generation. These methods are called during the visualization recommendation and generation stages. Furthermore, the added chart implementation must be registered in the `__init__.py` and `chartdict.py` configuration files, listing all the supported VizKG visualizations.

4 Evaluation

In this section, we report on the visualization recommendation evaluation and use case evaluation of VizKG.

4.1 Visualization Recommendation Evaluation

To evaluate the accuracy of VizKG visualization recommendation (as described in Section 3.2), we conduct an experimental evaluation using real-world SPARQL queries taken from the Wikidata query examples page (as of September 16, 2021).[14] We take only the visualization queries from the page, that is, those queries starting with the "#defaultView:[viz-type]" directive where viz-type refers to the preferred visualization type for the query results. We consider 82 out of 92 visualization queries because the remaining 10 queries either give no query results or a time limit error. We run the queries using our VizKG library, and record the visualization recommendations given by VizKG. Then, we check whether the VizKG recommendations include the original, preferred visualization type, as stated in the "#defaultView:[viz-type]" directive of the Wikidata queries.

From the experiment results, we observe that VizKG gives a correct recommendation in 72 out of 82 cases, giving an accuracy of 87.8%. One of the reasons for the incorrect recommendations is that there is a difference in the mapping rules between WQS and VizKG: WQS only requires one numeric variable to generate a scatter chart, whereas VizKG requires two numeric variables. Overall, we believe that the visualization recommendation of VizKG delivers quite a good result considering that the recommendation procedure (as described before) is fairly simple.

4.2 Use Case Evaluation

We highlight use cases of VizKG in several domains (i.e., COVID-19, cultural heritage in Indonesia, and higher education) over a number of knowledge graphs (i.e., Wikidata, DBpedia, BudayaKB, and data.open.ac.uk). The visualization results are discussed below and showcased in Fig. 2.

[13] https://github.com/fadirra/vizkg
[14] https://bit.ly/WD-queries-examples

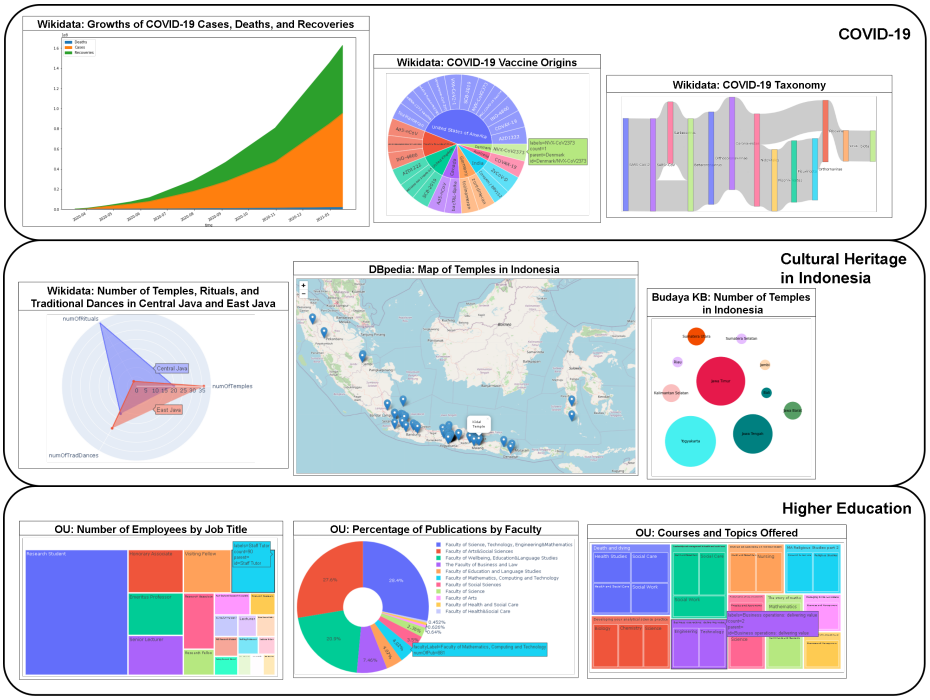


Fig. 2. VizKG Use Case Visualization (bit.ly/VizKG-Usecase)

COVID-19. This domain is relevant at the moment, particularly in regard to the latest updates of the COVID-19 pandemic. The data source used is Wikidata, accessed through its SPARQL endpoint.[15] The visualizations made are about: (i) the growth of COVID-19 cases, deaths, and recoveries using the stacked area chart type; (ii) origins of COVID-19 vaccines using the sunburst type; and (iii) COVID-19 taxonomy using the dimensions visualization type.

Cultural Heritage in Indonesia. In this domain, VizKG is evaluated over three different KGs: Wikidata, DBpedia, and BudayaKB. As a context, BudayaKB is a KG specialized for Indonesian cultural heritage, ranging from traditional music to folklores [10]. The left figure uses a radar chart to compare the number of traditional dances, rituals, and temples in the province of Central Java and East Java. The middle figure visualizes how Indonesian temples distribute on the map. The right figure illustrates the different number of temples by Indonesian provinces using the bubble chart type.

The middle figure, which is based on DBpedia,[16] relies on the following SPARQL query that retrieves Indonesian temples (= "candi" in Indonesian) including their geolocations and labels.

[15] https://query.wikidata.org/sparql
[16] https://dbpedia.org/

```
SELECT * WHERE {
     ?item dbo:wikiPageWikiLink dbr:Candi_of_Indonesia;
           geo:geometry ?geo .
     ?item rdfs:label ?itemLabel.
     FILTER(LANG(?itemLabel) = "en")
}
```

In the query, there are three variables (in the order they appear): `?item`, `?geo`, and `?itemLabel`. Via the VizKG mapping rules (as mentioned in Section 3.2), the query results can be visualized into a map, by mapping the `?geo` variable to the coordinate component and the `?itemLabel` variable to the popup component of the visualization. Note that `?item` need not be mapped in this case. A video demonstrating how VizKG can be used to visualize a map of Indonesian temples in DBpedia is available.[17]

Higher Education. Here, the data is obtained from the Open University (OU).[18] The topics are about employee profiles, courses, and publications of the Open University. We showcase the number of employees based on the job title with the treemap visualization, the number of publications by faculty using the donut chart type, and offered topics and courses using the treemap.

5 Conclusions and Future Work

This paper proposes VizKG as a Python-based framework for visualizing SPARQL query results over KGs. The framework facilitates the creation of visualization for generic KGs, automatic visualization recommendation, and extension mechanism for new visualization types. The VizKG visualization recommendation has been evaluated over real-world queries from Wikidata and achieves an accuracy of 87.8%. The use case evaluation of the VizKG framework is done over three domains (i.e., COVID-19, cultural heritage, and higher education) and four KGs (i.e., Wikidata, DBpedia, BudayaKB, and Open University). All of our use case examples are available as a Google Colab notebook.[19]

For future work, we plan to enhance the visualization recommendation with compatibility ranking so that not only binary recommendation is given. We also envision that more rigorous user testing can be performed to get a better understanding of the features and limitations of the VizKG framework. Furthermore, integrating the VizKG framework to kglab,[20] a hub package for graph-based data science libraries, can be a viable future direction to support wider audience.

[17] https://bit.ly/VizKGDemoTemples
[18] https://data.open.ac.uk/
[19] https://bit.ly/VizKGColab
[20] https://derwen.ai/docs/kgl/

Acknowledgements

We thank the anonymous reviewers for their careful feedback. The dissemination of this work is funded by a grant from Program Kompetisi Kampus Merdeka (PK-KM) 2021 of Faculty of Computer Science, Universitas Indonesia.

References

1. Atemezing, G.A., Troncy, R.: Towards a Linked-Data based Visualization Wizard. In: COLD (2014)
2. Brunetti, J.M., Auer, S., García, R., Klímek, J., Nečaskỳ, M.: Formal Linked Data Visualization Model. In: IIWAS (2013)
3. Ehrlinger, L., Wöß, W.: Towards a Definition of Knowledge Graphs. In: SEMANTiCS Posters and Demos (2016)
4. Fayyad, U.M., Grinstein, G.G., Wierse, A. (eds.): Information Visualization in Data Mining and Knowledge Discovery. Morgan Kaufmann (2001)
5. Gómez-Romero, J., Molina-Solana, M., Oehmichen, A., Guo, Y.: Visualizing Large Knowledge Graphs: A Performance Analysis. Future Gener. Comput. Syst. **89**, 224–238 (2018)
6. Heer, J., Bostock, M., Ogievetsky, V.: A Tour through the Visualization Zoo. CACM **53**(6), 59–67 (2010)
7. Heim, P., Lohmann, S., Tsendragchaa, D., Ertl, T.: SemLens: Visual Analysis of Semantic Data with Scatter Plots and Semantic Lenses. In: I-Semantics (2011)
8. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In: ISWC (2018)
9. Perkel, J.M.: Why Jupyter is Data Scientists' Computational Notebook of Choice. Nature **563**(7729), 145–146 (Oct 2018)
10. Putra, H.S., Mahendra, R., Darari, F.: BudayaKB: Extraction of Cultural Heritage Entities from Heterogeneous Formats. In: WIMS. pp. 6:1–6:9 (2019)
11. Ramadhana, N.H., Darari, F., Putra, P.O.H., Nutt, W., Razniewski, S., Akbar, R.I.: User-Centered Design for Knowledge Imbalance Analysis: A Case Study of ProWD. In: VOILA (2020)
12. Rietveld, L., Hoekstra, R.: YASGUI: Not Just Another SPARQL Client. In: SALAD (2013)
13. Ristoski, P., Paulheim, H.: Semantic Web in Data Mining and Knowledge Discovery: A Comprehensive Survey. J. Web Semant. **36**, 1–22 (2016)
14. Thellmann, K., Galkin, M., Orlandi, F., Auer, S.: LinkDaViz – Automatic Binding of Linked Data to Visualizations. In: ISWC (2015)

Displaying triple provenance with extensions to the Fresnel vocabulary for semantic browsers

Lloyd Rutledge[0000−0003−2814−9483], Pascal Mellema,
Tije Pietersma, and Stef Joosten

Open University of the Netherlands, Heerlen, The Netherlands
Lloyd.Rutledge@ou.nl

Abstract. We propose extensions to the Fresnel semantic browser vocabulary that displays data about triples instead of just about resources. This facilitates broadly applicable rapid prototyping of information systems that include displaying the provenance of triples. This provenance can include a triple's role in logical conclusions, how it can be edited and its documented sources. Our contribution's technical focus is the addition of a box for reification to Fresnel's display model. We also propose extensions to SPARQL selectors in Fresnel for querying whole triples as context instead of single resources. We evaluate our extensions with screen displays within an illustrative scenario that applies them. Furthermore, we demonstrate our proposal's technical feasibility with prototype software that generates these displays: a component called TransFresnel (Transparent Fresnel) in a semantic browser called RuleStyle.

Keywords: Provenance · Justification · Fresnel · Semantic browsers.

1 Introduction

Users increasingly demand from their Semantic Web interfaces not only data but also its origins. Such provenance often includes source documents, justification of conclusions, and interfaces for editing that data. This enables users, for any given unit of data they browse upon, to check if it is correct and, when needed, correct it. Of these types of provenance, presenting explanations of reasoning to end users is the primary motivating application for this work. It is also the subject of much recent research [16].

The Fresnel vocabulary for semantic browsers offers a generalized technology for designing data interfaces to Semantic Web data [11]. It takes a presentation-independent stylesheet-based approach similar to that of CSS. Thus, one Fresnel stylesheet can apply to multiple ontologies, such as one CSS stylesheet can apply to HTML from different websites. In addition, one ontology can be presented in different ways by different Fresnel stylesheets. This enables efficient creation of reusable stylesheets for browsing unfamiliar datasets in a tailored and readily adaptable style and manner.

In earlier work, we explored applying Fresnel to making rapid prototypes of basic end-user information system interfaces for data in new ontologies [14]. We implemented this in the software Fresnel Forms. Fresnel Forms automatically generates

a basic, default Fresnel stylesheet for any given ontology. In addition to generating stylesheets, this software processes Fresnel stylesheets into user interfaces that offer not just data browsing but also form-based data input. These forms also provide some basic implementation of business rules by, for example, setting logic-based constraints on what data users can enter. Therefore, Fresnel Forms shows how Fresnel can provide a rapid prototype of a basic business information system for any ontology.

While, until now, Fresnel has only been applied to displaying information about individual resources, we propose here an extension to Fresnel for specifying the display of information about triples instead of just about instances. This Fresnel extension facilitates rapid prototyping of business information systems that provide transparency for business rule logic along with the more general data. We use the Protégé ontology editor's visual interface as an example to emulate for displaying reasoning in the Semantic Web along with its justification [7]. Validation comes in part from illustrative scenarios of this approach applied to the fictional business rule example EU-Rent [8]. This work also validates its proposals in prototype software we name TransFresnel (Transparent Fresnel), as a component in the broader Fresnel-conformant semantic browser RuleStyle.

2 Related Work

This section presents related work in the key relevant subject areas. Triple provenance display is the functionality we aim to introduce into our proposed solution. Protégé provides interface functionality that our more generalized approach should be able to emulate. EU-Rent is the often-used fictional case we validate our proposal with. Finally, Fresnel is the technological foundation for our proposed solutions. The related work this section presents includes existing ontologies that apply to our work's semantic needs. We aim to maximize reuse of such ontologies by using them as is where possible and extending them where necessary.

2.1 Justification and other provenance

The primary technology for provenance on the Semantic Web is reification: treating a triple as a resource so it can be the subject of other triples. Its original technical specification is RDF's `Statement` class as a domain for the properties `subject`, `predicate` and `object`. In addition, there are proposals for other arguably more efficient representations of reification than requiring four additional triples to make each triple reifiable. RSP-QL*, for example, offers both RDF and SPARQL the syntactic shortcut of encapsulating simple triple expressions in double angle brackets [5]. This work here proposes how Fresnel can recognize and display reification as encoded by a variety of ontologies for it.

The type of provenance that motivates this work is the explanation of the reasoning behind specific triples. The Semantic Web Application Platform's (SWAP) reason ontology is a "vocabulary for proofs"[2]. Its namespace offers the class `Inference`

[2] https://www.w3.org/2000/10/swap/reason

for inferences, the property `gives` for the triples that are inferred, and the property `evidence` for the triples causing the inference. As such, SWAP's reason ontology directly supports justification of inferenced triples. We apply this ontology and these components here in our examples of how our proposal can have Fresnel present justifications of inferred triples.

The tool Validatrr applies these SWAP constructs in explanations it generates for SHACL constraint violations [3]. It does so by having the object of the `gives` property not be a single inferred triple, but instead by treating each violation as inferred, and thus giving the collection of inferred triples defining that violation. We propose here how Fresnel can select such inferences and display the justifications for them. Validatrr reifies triples with RDF containers of three resources instead of using RDF reification constructs. Fresnel SPARQL queries can query this construction of reification, and our examples in this paper do so.

Proof Markup Language (PML) represents justifications for data that Semantic Web services produce, such as inferencing [2]. PML broadened its scope to more general provenance of information and had large influence on the development of the PROV ontology for provenance [1].

In earlier work, we proposed design patterns for inferences from which reasoners generate understandable justifications [10]. Then, we used Protégé to process example code to generate illustrative scenarios. We now apply Protégé's reasoner and explanation boxes in the same manner, using the same illustrative scenario evaluation technique. More recently, we proposed a reference architecture for implementing traceability in rule-based information systems [15]. Such traceability is another form a provenance. The techniques we propose now can apply to implementing parts of that reference architecture.

2.2 Protégé

The Protégé ontology editor [7] is often used in Semantic Web research and industry. We apply Protégé screendumps here in illustrative scenarios. The focus is on Protégé's facilities for highlighting and explaining inferred triples.

Fig. 1 shows a Protégé display with a triple highlighted as inferred. Protégé shows inferred triples by giving their display a distinct style: a yellow background. In previous work, we used such Protégé displays as examples to emulate with Fresnel [6]. This current work does so as well. However, this earlier work proposed a Fresnel selector as a binary property stating whether a given triple is inferred. This current work, on the other hand, proposes extending selector queries to include contextual triples along with provenance data about them.

Fig. 1 also shows a Protégé display with an explanation for an inference. This work uses these displays to show what information an explanation offers. We then show how letting selector queries access such information helps with styling inferred triple display and showing explanation information.

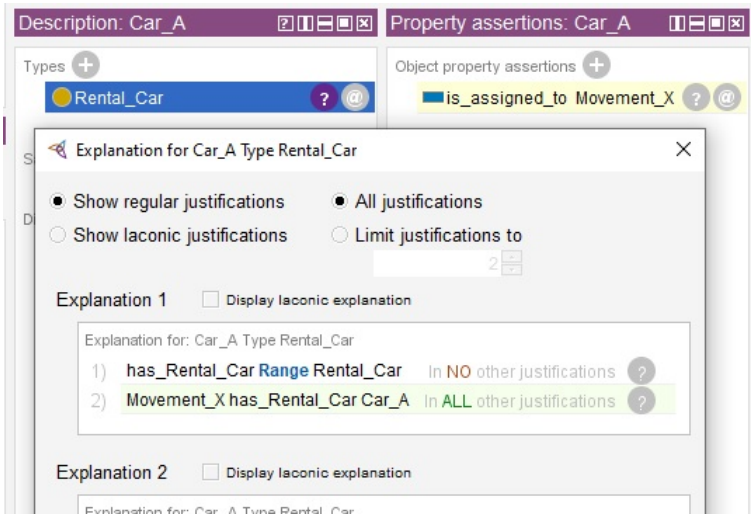


Fig. 1. Protégé display of inferred triples. One is indicated as inferred with a yellow background. An explanation box shows for another inferred triple.

2.3 EU-Rent

EU-Rent is an informative example of the business rule notation format Semantics of Business Vocabulary and Business Rules (SBVR) [8]. It provides SBVR code for the data model and business rules that define how a fictional car rental company runs. Other research has since applied EU-Rent to illustrate and evaluate other business rule formats and technologies as well. In one such work, Reynares uses the EU-Rent case as example application of a proposed mapping from SBVR to OWL-2 [12]. This section later presents our previous work on implementing business rules with Semantic Web and Fresnel technologies that uses EU-Rent in this way. We also discussed then how this previous work has built on the OWL ontology that Reynares wrote for EU-Rent. This paper here continues with EU-Rent in general, and Reynares's OWL ontology in particular, as an example for displaying triple metadata with Fresnel stylesheets.

2.4 Fresnel

Fresnel is a Semantic Web vocabulary for defining stylesheets for how browsers display Semantic Web data to end-users [11]. Its general model is that, at each step of browsing, the interface displays all triples for the current subject's resource in a table. The user can then click on objects for the resources displayed for a current subject to navigate to the display with that resource as the new current subject. Typically, a table for the Fresnel display of a resource shows a triple in each row with columns for the predicates and objects.

Fresnel's box model lets stylesheet specifications apply to either the whole table, or the row for a property, or the cell for a property or object. Fig. 2 shows Fresnel's box

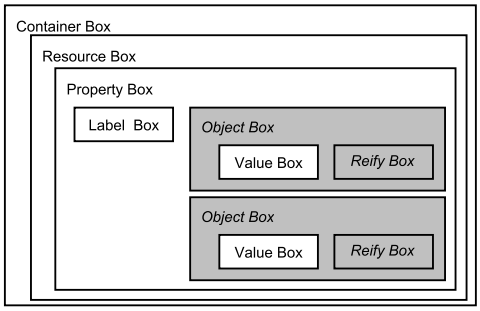


Fig. 2. The box model for Fresnel displays [11], with our extension to it in italics and gray backgrounds.

model for its browser displays [11], along with an extension to it that we propose later in this paper. The container box in this model represents an entire display at a given interaction moment on a Fresnel browser. It contains the resource box, which contains the display for all data regarding a given Semantic Web resource. Typically, this display shows the triples for which the given resource is the subject. The resource box contains property boxes, which each display a label for the property and all the objects in triples for this property and the current subject. The label box contains the label of the given property. Then the value box shows the object for a triple. Later, this paper describes the object and reify boxes that we propose as Fresnel extensions.

Each style specification, or "lens", in Fresnel has a selector that determines which resources' displays to apply its style to. The most general type of Fresnel selector is a SPARQL query. The URIs that a lens's selector query returns are the resources that the browser applies the lens's styles to.

Fresnel's format domain properties all have selectors triggered by a single resource, be it an individual, class or property. Property format domains, on the other hand, affect the style of the cells for a given property in any table using it. If such a selector is a SPARQL query then the given style applies to the display of the current resource if the list returned contains the resource. As such, Fresnel is, up until now, resource-centric: selectors query for resources to decide how to display all triples involving that resource. We describe later how our proposed extension to Fresnel is necessarily triple-centric because the style is determined by a triple and applies to the display of that triple in the context of the broader display of all information around a resource.

2.5 Extending Fresnel beyond data browsing

Fresnel has typically focused on resource-driven browsing of existing, non-changing data. In earlier work, we explored extending Fresnel to define interfaces for form-based addition of new data by the user, implemented in software called Fresnel Forms [14]. Fresnel Forms also provides the automated generation of default stylesheets for any

given ontology. These features combined to have Fresnel Forms provide rapid prototyping of information systems by generating a browse and data input interface for any ontology.

We then zoomed in from information systems to business systems by proposing another extension to Fresnel for handing business rule violations that are triggered by data the user enters [13]. With this approach, a Fresnel stylesheet can determine whether the business system handles violations of a given rule by blocking entry of the data into the system, or by allowing it but warning the user of the violation. Thus, the block and the warning are two different styles that can apply to the same business rule. We apply a scenario from EU-Rent to illustrate and evaluate this technique.

Later work of ours has proposed extensions to Fresnel that directly specify style for presenting inferred triples, again using EU-Rent as an example [6]. Now we build upon that work by presenting a more general extension to Fresnel that enables the same styling for provenance. EU-Rent continues to apply here as well as a source for illustrative scenarios. This paper goes further by showing explanations for inferences, styling and explanations for reasoning beyond inferencing, and displaying data for triples beyond reasoning about them.

3 Method

The methodology the rest of this paper applies in its research has three components, each in each of the following two sections: specifications and validation. First, we propose new technological components for the Fresnel vocabulary in the form of their specification. These focus on a new box in the Fresnel box model that displays information about triples, as well as on how content is selected for that box and which style applies to it. This simple extension aims to reuse as much of existing Fresnel as possible and thus be minimal. Where extension to Fresnel remains nonetheless necessary, its technical form will be as consistent as possible with the rest of Fresnel.

Then we validate this extension with the design science evaluation techniques of illustrative scenarios and prototyping [9]. The scenarios implement common, existing types of reification displays. They show how one can encode Fresnel with this extension to display reification along with resource data. We present these illustrative scenarios as browser displays that demonstrate the functionality of each extension. These displays accompany the fragments of Fresnel code that generate them, which include our extensions and examples. We create data for these scenarios as instances of EU-Rent's conceptual model and business rules EU-Rent [8] as encoded in OWL by Reynares [12].

We also evaluate our proposal by developing prototype software for it: the TransFresnel component of our RuleStyle browser. TransFresnel generates the illustrative scenario displays mentioned above by processing the code we present with them. The scenarios are thus not only illustrative; our prototype implementation that generates them supports the technical feasibility of our approach.

4 Specifications

This section presents technical specifications of our proposed extension to Fresnel that account for reification triples. There are two types of specifications: new vocabulary components, and new abstract model components to which the vocabulary components apply. We use the namespace prefix `transfr:` (TRANSparent FResnel) for new constructs that we propose adding to Fresnel.

In Fresnel, formats encapsulate style definitions that apply to components of the box model in a display. Each format can have format domain properties. These define the domains, or patterns of displayed semantic data, to which each format applies. Fresnel has format domains for properties, classes and instances. The latter two apply to resource boxes in the box model by selecting the resources to display there. Property format domains, on the other hand, specify which properties receive the given style in a property or label box. Each current Fresnel format domain regards a single URI: that of a class, instance or property. The extensions we propose ahead in this section each addresses the need of applying style to the display of a triple and its provenance instead of a single resource.

4.1 Reify and object boxes

Fig. 2 shows the current Fresnel box model along with the extension we propose to it. This extension is the addition of two boxes to the model: the object box, and the reify box. The reify box is the core of this extension. It comes just after the value box in the Fresnel box model.

A reify box is similar to a resource box in that it displays triples with a given subject. The key difference is that, while this subject is typically a single resource for a resource box, for a reify box, the subject is itself a triple. Furthermore, while a resource box is typically an entire, independent display, a reify box is linked to a value box, joined in an object box, that is itself a cell, or item in a cell, within a larger tabular resource box.

The reify box is a place where a Fresnel stylesheet can place information from reifications for the triple associated with the adjacent value box. When using the example display of Protégé from Fig. 1 as an analogy, the reify box would be where the buttons with the '?' and '@' icons are. The object box in our extension supports the reify box by keeping it attached to the value box for the triple it describes. It also allows the value and reify boxes to be styled together as all information about the given triple.

4.2 SPARQL selectors with triples as context

The coming subsections propose new format domains for Fresnel. Since Fresnel's previous format domains all have individual resources as context, their SPARQL selectors each return a single variable binding. When this matches a resource, the resource falls in the domain and is thus displayed with the associated style.

Our new format domains' types of triggers no longer have single resources as context but instead entire triples. This can affect SPARQL selectors in two ways. One is that the SPARQL query returns not one but three variable bindings, which then must

match the three URIs in a triple. The other effect is our addition of three prebound variables that let SPARQL query with the given triple as reference context. The subsections ahead describe how each of these is used.

4.3 transfr:valueFormatDomain and transfr:objectFormatDomain

In Fresnel's current box model, the value box is what corresponds best to the display of a triple. The value that each Fresnel value box shows is the object in a triple with the content of the preceding property box as the property and the resource of the encapsulating resource box as the subject. While Fresnel provides the value box for displaying triples, it provides no format domain for it, and thus no means of tailoring the style of the value it presents based on its triple.

Here we propose the property `transfr:valueFormatDomain` as an extension for Fresnel. This specifies which triples have the presentation style of the given value box. It applies as well to the object and reify boxes we propose in this work. SPARQL selector queries that value format domains use return not one but three variables, which correspond to the three URIs of the context triple displayed in a given value box.

Our extension adds an object box that encapsulates Fresnel's value box with our extension's reify box. To be able to format both the value and object boxes together through their object box, we also propose extending Fresnel with the `transfr:objectFormatDomain` property. This works for object boxes equivalently to how `transfr:valueFormatDomain` works for value boxes.

4.4 transfr:reifyFormatDomain and prebound triple variables

In our proposed additions to Fresnel, the property `transfr:reifyFormatDomain` specifies the format domain for the reify box. As with our value format domain, we propose some unique characteristics of SPARQL selector queries for reify format domains. One is that it returns a single variable, instead of the three variables for the reified triple by value format domains. Furthermore, this single variable is a resource for reification information for the triple, rather than for the context to be matched. The URI bound to this single returned variable lets the browsers put a link to it in the reify box.

In order to let SPARQL selector queries in reify format domains still match context reified triples, we also propose adding three prebound variables to represent the triple. These are: `$thisSubject`, `$thisPredicate` and `$thisObject`. They are replaced by the subject, predicate and object of the triples being displayed in such SPARQL queries. In this way, the queries get not just the context of triples to apply their style to, but they can be tailored to return resources containing reifications for the triples.

4.5 transfr:reifyStyle and transfr:reifyLabel

Fresnel provides what it calls styling hooks for each box in its model. Each type of box has gets a hook as a property that assigns a format a string of CSS code for styling it. Since our extension to Fresnel adds a new box and corresponding format domain

for reifications, we add a styling hook for them as well. This extension is the property `transfr:reifyStyle`, which applies to the reify box. We also proposed adding the property `transfr:reifyLabel` for reify formats, which lets a string of text serve as the content of reify boxes styles by the formats. It is based on the existing property `fresnel:label` for property formats, which defines the property-describing text content of label boxes. In addition, this text serves as a link to the browser lens display for the reification.

5 Validation scenarios and prototype

This section evaluates our proposed technical specifications from the previous section by applying them to implement the EU-Rent case and emulate its display on Protégé. It uses EU-Rent for illustrative scenarios to which to apply the Fresnel extensions. We also present a prototype for these extensions that generates displays for these scenarios.

5.1 Style: Yellow background for inferred triples

There are several ways one could encode the inclusion of explanation triples in what Fresnel selectors query. Fig. 3 shows the display from our browser RuleStyle highlighting an inferred triple. Fig. 3 is our recreation of the Protégé display in Fig. 1 with the same Semantic Web code processed instead with RuleStyle. Fig. 4 shows how we set Protégé's style for inferred triples in Fresnel.

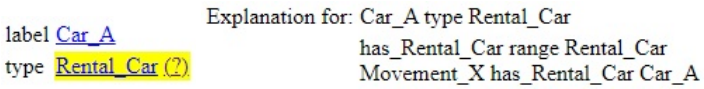


Fig. 3. Prototype displays for the style in Fig. 4 and the explanation in Fig. 6.

```
ex:infFormat rdf:type fresnel:Format ;
 transfr:objectFormatDomain """
  SELECT ?thisSubject ?thisPredicate ?thisObject
  WHERE { ?Inferred a reas:Inference ;
   reas:gives/rdf:first                  ?thisSubject   ;
   reas:gives/rdf:rest/rdf:first         ?thisPredicate ;
   reas:gives/rdf:rest/rdf:rest/rdf:first ?thisObject     .
  }"""^^fresnel:sparqlSelector ;
fresnel:objectStyle "background-color:yellow"^^fresnel:stylingInstructions .
```

Fig. 4. Fresnel code with CSS code emulating Protégé's styling of inferred triples.

5.2 Links: Question mark link to justification display

Fig. 5 shows how we put a question mark as an anchor to explanations for triples in Fresnel. The tiny string with the question mark can be replaced with others for other types of reifications, such as with square brackets to represent citations.

```
ex:reifLinkFormat rdf:type fresnel:Format ;
 transfr:reifyFormatDomain """
  SELECT ?Inferred WHERE { ?Inferred a reas:Inference ;
   reas:gives/rdf:first                  $thisSubject   ;
   reas:gives/rdf:rest/rdf:first         $thisPredicate ;
   reas:gives/rdf:rest/rdf:rest/rdf:first $thisObject     .
  }"""^^fresnel:sparqlSelector ;
 transfr:reifyLabel "(?)"^^xsd:string .
```

Fig. 5. Fresnel code emulating Protégé's links to explanation boxes.

5.3 Explanation boxes

Fig. 6 shows Fresnel code for the illustrative scenario for displaying justification triples. Here, the query applies to data using the SWAP ontology as implemented in Validatrr [3]. It selects members of the `Inference` class.

5.4 Prototype

The code for the software and the demonstrations we present here for it are available online[1]. This prototype software is the partial Fresnel browser RuleStyle, which we started as a proof of concept and feasibility study for this paper. It generates the screen displays shown in Fig. 3. It validates the scenarios presented here by implementing them, showing that a program can have the scenarios function as shown. We have thus implemented in RuleStyle no more than is needed to generate the scenario screen displays from the code in this paper.

RuleStyle is a PHP server script that references a SPARQL endpoint to collect and display data regarding the given resource. The SPARQL endpoint we set up for our installed RuleStyle server is Jena Fuseki. We put the N3 scenario code in a larger N3 file with contextual code, which we then convert to XML RDF for Jena. This contextual code includes fragments of the EU-Rent ontology and our small test population for it. It also includes inferences from this population code generated by Protégé. The explanation code is typed in by hand from Protégé explanation box displays.

As of this writing, RuleStyle provides generalized implementation for Fig. 4's example of the styled display of inferred triples. For the link implementation in Fig. 5,

[1] https://github.com/LloydRutledge/RuleStyle

RuleStyle retrieves and presents the icon in a generalized fashion. However, the pre-bound variables in the selector still need implementing. Finally, RuleStyle supports the selection of the explanation box lens in Fig. 6.

```
ex:explBox a fresnel:Lens ; fresnel:instanceLensDomain """
  SELECT ?inference
  WHERE {?inference a reas:Inference}"""^^fresnel:sparqlSelector ;
 fresnel:showProperties (
  [ fresnel:property reas:gives    ; fresnel:sublens ex:stmtLens ]
  [ fresnel:property reas:evidence ; fresnel:sublens ex:evidLens ] ) .

ex:stmtLens a fresnel:Lens ;
 fresnel:showProperties ( rei:subject rei:predicate rei:object ) .

ex:evidLens a fresnel:Lens ;
 fresnel:showProperties ( rdf:first
  [ fresnel:property rdf:rest ; fresnel:sublens ex:evidLens ] ) .
```

Fig. 6. Fresnel code for displaying an explanation box.

6 Conclusion

This paper adds triple provenance display to Fresnel by extending its box model and SPARQL selectors. The new reify box can contain links to reification information for triples displayed. SPARQL selectors can now acknowledge triples as context instead of just single resources, and return reification information links for contextual triples. This work builds upon the Master's theses of Pascal Mellema [6] and Tije Pietersma [10].

This work represents several iterations of a design science approach. Additional potential extensions to Fresnel these iterations motivate include having the reify box selectors return a property with the object, and extending `fresnel:showProperties` to include property paths, perhaps in the form of semantic paths [4]. Having reify box selectors return an additional variable binding for a property along with that for the object it links to would let the browser know what type of provenance it provides. This would enable adapting our explanation link example to present other icons for other types for provenance, such as Protégé's '@' for annotation, brackets for citations such as in Wikipedia infoboxes, and links for editing the data.

In addition, Fresnel's `fresnel:showProperties` list currently only contains single URI's. We would like to extend this to include SPARQL property paths as well. This would facilitate presenting SWAP triples because its members are in an RDF list. Property paths could locate each triple member in the list so our extended Fresnel treats is as a property of the triple and displays it as such.

Other potential future work for which this new technology lays a foundation includes generating human readable explanations. To do this, specialized lenses can select justification triple sets that match readable text templates with SPARQL queries

for given shapes. Finally, browsers such as Rule Style that implement this provenance-based extension to Fresnel can apply it in broader business information system research.

References

1. PROV-Overview. An Overview of the PROV Family of Documents. Project report (April 2013), https://eprints.soton.ac.uk/356854/
2. da Silva, P.P., McGuinness, D.L., Fikes, R.: A proof markup language for Semantic Web services. Information Systems **31**(4), 381 – 395 (2006). https://doi.org/https://doi.org/10.1016/j.is.2005.02.003
3. De Meester, B., Heyvaert, P., Arndt, D., Dimou, A., Verborgh, R.: RDF Graph Validation Using Rule-Based Reasoning. Semantic Web Journal (2020)
4. Destandau, M., Appert, C., Pietriga, E.: S-Paths: Set-based visual exploration of linked data driven by semantic paths. Semantic Web **12**(1), 99–116 (01 2021). https://doi.org/10.3233/SW-200383
5. Keskisärkkä, R., Blomqvist, E., Lind, L., Hartig, O.: RSP-QL: Enabling Statement-Level Annotations in RDF Streams, pp. 140–155 (11 2019). https://doi.org/10.1007/978-3-030-33220-4_11
6. Mellema, P.: Extending semantic browser style specifications for Semantic Web inferencing. Master's thesis, Open University of the Netherlands, Heerlen, The Netherlands (2020)
7. Musen, M.A., Protégé, T.: The Protégé Project: A Look Back and a Look Forward. AI matters **1**(4), 4–12 (2015). https://doi.org/10.1145/2757001.2757003
8. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR), Appendix G - EU-Rent Example (2016)
9. Peffers, K., Rothenberger, M., Tuunanen, T., Vaezi, R.: Design science research evaluation. In: Proceedings of the 7th International Conference on Design Science Research in Information Systems: Advances in Theory and Practice. p. 398–410. DESRIST'12, Springer-Verlag, Berlin, Heidelberg (2012)
10. Pietersma, T.: Ontologie-ontwerppatronen voor gevolgtrekkingen met automatische uitleg. Master's thesis, Open University of the Netherlands, Heerlen, The Netherlands (2019)
11. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A browser-independent presentation vocabulary for RDF. In: International Semantic Web Conference. pp. 158–171. Springer (11 2006)
12. Reynares, E., Caliusco, M., Galli, M.: SBVR to OWL 2 mappings: An automatable and structural-rooted approach. CLEI Electronic Journal **17** (12 2014). https://doi.org/10.19153/cleiej.17.3.2
13. Rutledge, L., Bouwer, E., Joosten, S.: Rule style: Patterns of and extensions to data system user interface specification for business rule violations. In: Shishkov, B. (ed.) Business Modeling and Software Design. pp. 3–16. Springer International Publishing, Cham (2019)
14. Rutledge, L., Brenninkmeijer, T., Zwanenberg, T., van de Heijning, J., Mekkering, A., Theunissen, J.N., Bos, R.: From ontology to semantic wiki – designing annotation and browse interfaces for given ontologies. In: Molli, P., Breslin, J.G., Vidal, M.E. (eds.) Semantic Web Collaborative Spaces. pp. 53–72. Springer International Publishing, Cham (2016)
15. Rutledge, L., Italiaander, R.: Toward a reference architecture for traceability in SBVR-based systems. In: Proceedings of the Seventh International Workshop on Controlled Natural Language (CNL 2020/21). Special Interest Group on Controlled Natural Language, Amsterdam, Netherlands (Sep 2021), https://aclanthology.org/2021.cnl-1.13
16. Waltl, B., Vogl, R.: Explainable artificial intelligence: The new frontier in legal informatics. Jusletter IT **4**, 1–10 (2018)