Proceedings of the 8th International Workshop on

Visualization and Interaction
for
Ontologies, Linked Data and Knowledge Graphs

Co-located with ISWC 2023, Athens, Greece, November 6, 2023.
Organizing Committee
Bo Fu, California State University Long Beach, USA
Patrick Lambrix, Linköping University, Sweden
Huanyu Li, Linköping University, Sweden
Susana Nunes, LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
Catia Pesquita, LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Program Committee
Karlis Čerāns, University of Latvia, Latvia
Christophe Debruyne, University of Liège, Belgium
Roberto García, Universitat de Lleida, Spain
Ali Hasnain, Royal College of Surgeon, Ireland
Mayank Kejriwal, University of Southern California, USA
Albert Navarro Gallinad, Trinity College Dublin, Ireland
Declan O’Sullivan, Trinity College Dublin, Ireland
Evan Patton, Massachusetts Institute of Technology, USA
Emmanuel Pietriga, INRIA, France
Harald Sack, FIZ Karlsruhe, Leibniz Institute for Information Infrastructure & KIT Karlsruhe, Germany
Ahmet Soylu, OsloMet – Oslo Metropolitan University, Norway
Markel Vigo, The University of Manchester, UK

Additional Reviewer
Tabea Tietz, FIZ Karlsruhe, Leibniz Institute for Information Infrastructure & KIT Karlsruhe, Germany
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, Linked Data and Knowledge Graphs 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, Linked Data and Knowledge Graphs.

Visualizing Ontologies, Linked Data or Knowledge Graphs 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 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 Semantic Web 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. 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 semantics, and proliferate the consumption of Ontologies, Linked Data and Knowledge Graphs.

These and related issues are addressed by the VOILA! workshop series concerned with Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs. The eighth edition of VOILA! was co-located with the 22nd International Semantic Web Conference (ISWC 2023) and took place as a half-day event on November 6, 2023. It was organized around scientific paper presentations and discussions.

The call for papers for VOILA! 2023 attracted 10 submissions in different paper categories.
Three reviewers were assigned to each submission. Based on the reviews, we selected 8 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 Heiko Paulheim and Bo Fu, the workshop chairs of ISWC 2023, for their continuous support during the workshop organization.

September 2023

Bo Fu,
Patrick Lambrix,
Huanyu Li,
Susana Nunes,
Catia Pesquita

VOILA! 2023
http://voila.visualdataweb.org/2023
## Contents

### Regular papers

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualizing Literary Linked Data for Public Library Users in the New User Interface for BookSampo – Finnish Fiction Literature on the Semantic Web</td>
<td>2</td>
</tr>
<tr>
<td>by Annastiina Ahola, Eero Hyvönen</td>
<td></td>
</tr>
<tr>
<td>Towards a UML-based notation for OWL ontologies</td>
<td>18</td>
</tr>
<tr>
<td>by María Poveda-Villalón, Serge Chávez-Feria, Sergio Carulli-Pérez, Raúl García-Castro</td>
<td></td>
</tr>
<tr>
<td>How to create easily a data analytic semantic portal on top of a SPARQL endpoint: introducing the configurable Sampo-UI framework</td>
<td>28</td>
</tr>
<tr>
<td>by Heikki Rantala, Annastiina Ahola, Esko Ikkala, Eero Hyvönen</td>
<td></td>
</tr>
</tbody>
</table>

### Short papers

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrating Sparklis and ViziQuer for Enhanced SPARQL Querying and Visualization</td>
<td>41</td>
</tr>
<tr>
<td>by Uldis Bojārs, Jūlija Ovčinnikova, Lelde Lāce, Artūrs Sprogis, Mikus Grasmanis, Kārlis Čerāns</td>
<td></td>
</tr>
<tr>
<td>Towards A Knowledge Graph-based Exploratory Search for Privacy Engineering</td>
<td>49</td>
</tr>
<tr>
<td>by Guntur Budi Herwanto, Fajar J. Ekaputra, Florina Piroi, Marta Sabou</td>
<td></td>
</tr>
<tr>
<td>A Method of Visual Presentation of Data Schemas</td>
<td>57</td>
</tr>
<tr>
<td>by Lelde Lāce, Aiga Romāne, Mikus Grasmanis, Kārlis Čerāns</td>
<td></td>
</tr>
<tr>
<td>Tree Visualization of Patient Information for Explainability of AI Outputs</td>
<td>63</td>
</tr>
<tr>
<td>by Sandeep Ramachandra, David Vander Mijnsbrugge, Pieter-Jan Lammertyn, Stijn Dupulthys, Femke Ongenae, Sofie Van Hoecke</td>
<td></td>
</tr>
<tr>
<td>Visualizing Ontology Metrics In The NEOntometrics Application</td>
<td>70</td>
</tr>
<tr>
<td>by Achim Reiz, Kurt Sandkuhl</td>
<td></td>
</tr>
</tbody>
</table>
Regular papers
Visualizing Literary Linked Data for Public Library Users in the New User Interface for BookSampo – Finnish Fiction Literature on the Semantic Web

Annastiina Ahola¹, Eero Hyvönen¹,²

¹Semantic Computing Research Group (SeCo), Aalto University, Finland
https://seco.cs.aalto.fi, firstname.lastname@aalto.fi
²Helsinki Centre for Digital Humanities (HELDIG), University of Helsinki, Finland

Abstract
The BookSampo Linked Data (LD) portal was deployed in 2011 by the Public Libraries of Finland and has today nearly 1.6 million annual users. Its large knowledge graph (KG) covers virtually all Finnish Fiction literature but has not been fully exploited in Digital Humanities. This paper discusses how the KG can be used for literary search, data exploration, and research by presenting a new BookSampo user interface (UI) based on faceted semantic search and browsing with seamlessly integrated data-analytic visualization tools. This application makes it possible for the first time to analyze the BookSampo data in versatile ways without programming skills. The analysis results presented suggest interesting spatial, temporal, and topical trends in how the Finnish fiction literature has evolved during the last decades.

1. Introduction

BookSampo¹ [1, 2] provides information on virtually all fiction literature published in Finland since mid 19th century. Its contents are based on rich semantic descriptions of books and their contexts using Linked Data (LD) that originates from multiple heterogeneous data sources. BookSampo is an application instance of the more general “Sampo Model”² for LD publishing and series of semantic portals in use³ in Finland and beyond [3].

BookSampo is used by library users and librarians for finding literary works of interest and related contextual information. The original Drupal-based UI in use since 2011 provides traditional text search engines for finding records and then related contents as links for data exploration [4]. However, the full potential of the underlying KG or searching, exploring, and for data analytic research has not been exploited.

To facilitate this, this paper presents a new semantic user interface (UI) for the BookSampo knowledge graph (KG). First in Section 2 the Sampo model underlying our work is overviewed.

²The model is called “Sampo” according to the Finnish epic Kalevala, where Sampo is a mythical machine giving riches and fortune to its holder, a kind of ancient metaphor of technology according to the most common interpretation of the concept.
After this the BookSampo KG is presented (Section 3). Based on using the Sampo-UI framework [5, 6], section 4 explains the new semantic portal with examples of using the system. In conclusion, related works are over-viewed, contributions of the new UI are discussed, and next steps ahead are outlined. This paper extends substantially our earlier short papers of the new project [7] and on using the BookSampo [8].

2. Sampo Model: Publishing and Studying Linked Data

Table 1
Sampo Model Principles P1–P6

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Support collaborative data creation and publishing</td>
</tr>
<tr>
<td>P2</td>
<td>Use a shared open ontology infrastructure</td>
</tr>
<tr>
<td>P3</td>
<td>Make clear distinction between the LOD service and the user interface (UI)</td>
</tr>
<tr>
<td>P4</td>
<td>Provide multiple perspectives to the same data</td>
</tr>
<tr>
<td>P5</td>
<td>Standardize portal usage by a simple filter-analyze two-step cycle</td>
</tr>
<tr>
<td>P6</td>
<td>Support data analysis and knowledge discovery in addition to data exploration</td>
</tr>
</tbody>
</table>

The Sampo model is a consolidated set of principles listed in Table 1 for collaborative LOD publishing and creating portals. The model is based on the Semantic Web standards [9] and best practices of the W3C for Linked Data publishing [10, 11].

Principles P1–P3 lay a foundation for developing LOD services. The model is based on the idea of collaborative content creation (P1). The data is aggregated from local data silos into a global service, based on a shared ontology and publishing infrastructure (P2). The model supports the idea of separating the underlying Linked Data service completely from the user interface via a SPARQL API (P3). This arguably simplifies the portal architecture and the data service can be opened for data analysis research in Digital Humanities.

The idea of principles P4–P6 is to “standardize” the UI logic of Sampo portals to be created on top of a LD service SPARQL endpoint. The goal is to make the portals easier to use and implement. Principle P4 articulates the idea of providing different thematic application perspectives by reusing the data service. They are used by a two-step cycle for research (P5): First the focus of interest, the target group, is filtered out using faceted semantic search [12, 13, 14]. Second, the target group is visualized or analyzed by using ready-to-use data analytic tools of the application perspectives. A novelty of the model is to support data analysis, visualization, and knowledge discovery with seamlessly integrated tooling, even solving research problems using AI [15] (P6).

To create Sampo portals, the Sampo-UI framework [5] has been designed and implemented. It has been used for developing all ca. 15 Sampo systems since 2018, including the new BookSampo portal, suggesting practical feasibility of the Sampo model [6].

---

4The name “Sampo” comes from the Finnish epic Kalevala, where Sampo is a mythical machine giving riches and fortune to its holder, a kind of ancient metaphor of technology.

5https://www.w3.org/standards/semanticweb/

6In our case the collaborators are institutions rather than individual people.
3. BookSampo Knowledge Graph

The user interface (UI) on top of the underlying knowledge graph (KG) in a SPARQL endpoint has been created using traditional search and data exploration methods, and the full potential of the KG—nearly 9 million triples today—has not been fully utilized: the data covers all Finnish fiction literature and beyond and is interesting from a Digital Humanities (DH) research perspective, too. Table 2 lists the number of instances of different entity types in the data from back in 2013 and today.

<table>
<thead>
<tr>
<th>Class (Type)</th>
<th>Instances</th>
<th>Class (Type)</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literary works</td>
<td>93,000</td>
<td>Literary works</td>
<td>215,000</td>
</tr>
<tr>
<td>Editions</td>
<td>127,000</td>
<td>Editions</td>
<td>222,000</td>
</tr>
<tr>
<td>Book covers</td>
<td>27,000</td>
<td>Book covers</td>
<td>119,000</td>
</tr>
<tr>
<td>Fictional characters</td>
<td>19,000</td>
<td>Fictional characters</td>
<td>49,000</td>
</tr>
<tr>
<td>Contemporary reviews</td>
<td>15,000</td>
<td>Contemporary reviews</td>
<td>15,000</td>
</tr>
<tr>
<td>Literary series</td>
<td>2,900</td>
<td>Literary series</td>
<td>8,900</td>
</tr>
<tr>
<td>Literary awards</td>
<td>2,700</td>
<td>Literary awards</td>
<td>6,400</td>
</tr>
<tr>
<td>Literary award series</td>
<td>200</td>
<td>Literary award series</td>
<td>300</td>
</tr>
<tr>
<td>People (e.g. authors)</td>
<td>29,000</td>
<td>People (e.g. authors)</td>
<td>64,000</td>
</tr>
<tr>
<td>Author’s pictures</td>
<td>2,600</td>
<td>Author’s pictures</td>
<td>4,200</td>
</tr>
<tr>
<td>Publishers</td>
<td>2,600</td>
<td>Publishers</td>
<td>5,500</td>
</tr>
</tbody>
</table>

Table 2
Instance counts in 2013 [1] (left) vs. 2023 (right)

Figure 1: An example of how a novel is modeled in the BookSampo KG.

Fig. 1 illustrates how the novel *Pride and prejudice* is modeled in the BookSampo KG. The
kaunokki:romaani entity represents the abstract work level of the novel. That entity has links to the entity of the author Austen, Jane and the kaunokki:fyysinen_teos entity representing the Finnish edition of the work translated by Joutsen, O. A. This Finnish edition further has the link to the publisher entity of the WSOY publishing house.

Figure 2: The abstract and physical work levels for Mika Waltari’s novel Sinuhe Egyptiläinen.

The BookSampo data divides works into two levels: abstract and physical work levels. The data model is based on the FRBRoo model [16] but simplified to the aforementioned two levels. The abstract work level is equivalent to the work level in the FRBRoo model while the physical work level represents what would be the manifestation level in the FRBRoo model. In practice the abstract work level deals with information that is shared between all editions of a work. The physical work level on the other hand contains edition-specific information, e.g., number of pages, and publisher, that are specific to the edition. In the case of the BookSampo data this edition-specific information is often recorded for the first editions in all relevant languages. That is to say, an average work will have a physical work level entities for at least its first edition in the original language as well as for the possible first editions of translations of the work into Finnish and/or Swedish. If a work is translated again later in time, an additional physical work level entity might be added for the new updated translation. Fig. 2 illustrates the split for Mika Waltari’s novel Sinuhe Egyptiläinen.

4. New BookSampo User Interface

This section first present the standard model of Sampo-UI for designing and implementing semantic portals. After this, it is shown how the model was used in the BookSampo portal.

**Generic Sampo-UI model** The new BookSampo User Interface (UI) is built using the Sampo-UI framework [5] based on the Sampo model [3] introduced in Section 2. A Sampo-UI interface contains a landing page with a set of “application perspectives” to choose from as depicted in Fig. 3. Each perspective provides the end-user with a faceted semantic search view to filter out individuals of a class (e.g., Novel, Person, Place, etc.) related to the perspective. After finding a set on instances (individuals) of interest, the user can either study and analyze them 1) one by one, 2) analyze sets of individuals together (e.g., statistically, on maps, or on timelines), or 3) explore the data by browsing based on internal links within the portal or links to external data sources such as Wikipedia or other Sampo portals. Data-analytic view and visualization can be customized for each application perspective for groups of instances and individual instances.
Figure 3: Navigational page structure of a portal based on Sampo-UI.

separately. From all pages links to a user guide, feedback channel to developers, and a page about the portal are provided (cf. the upper right corner in the figure).

A Sampo portal is implemented by configuring the pages types depicted in Fig. 3, i.e., the landing page, application perspective pages, and the instance pages. This is done using JSON (JavaScript Object Notation) configuration files [6]. The Sampo-UI framework offers ready-to-use components to be used in portal pages that can be added through the configuration files without the need for heavy coding. The components can easily be expanded upon by adding new mapping functions and expanding upon configuration options passed to the components.

Structure of the UI The user of BookSampo first lands on the landing page (shown in Fig. 4) when opening the portal. The portal consists of five application perspectives based on corresponding classes:

1. Novels. This perspective deals with the abstract work level of novels.
2. Publications. This perspective deals with the physical work level of all works.
3. People. This perspective deals with authors and other people related to literature, e.g., illustrators, translators, and reviewers.
4. Covers. This perspective deals with book covers and information related to them.
5. Nonfiction books. This perspective deals with the abstract work level of nonfiction books.

All of the perspectives query the data from the same SPARQL endpoint of BookSampo KG. Selecting a perspective represents the data from that perspective. For example, choosing the
**Figure 4**: The landing page of the BookSampo UI with five application perspectives.

The Novels perspective shows the data through novels. The search view of that perspective lists all novel entities in a list with links to the rest of the KG as their properties.

The basis for choosing the aforementioned perspectives was to cover all aspects of Finnish literature as comprehensively as possible with few perspectives. Novels perspective was chosen as the perspective to cover fictional books due to novels being the largest subgroup of fictional literary works in the data. Nonfiction books was chosen to supplement the Novels perspective with the nonfictional works, although in the data it only represents a non-comprehensive subset of nonfiction published in Finland. The Publications perspective covers all literary works on the physical work level. The split between 1) novels and nonfiction books and 2) publications follows the split made in the original BookSampo data [1] as introduced in Section 3.

To supplement the data on literary works, the People perspective was added to provide information on all people relevant to the presented data on literary works, whether it be the authors behind the books or other people relevant to them like the illustrators and translators of works. As the BookSampo KG also includes data on contemporary reviews, information of reviewers is also included in this perspective. To finish off the available perspectives, the Covers perspective was added due to the popularity of the book cover search function on the original BookSampo Portal, the search capabilities of which could be even further improved with the use of faceted search.

Clicking on a card for an application perspectives leads to that particular perspective’s **faceted search view** (see Fig. 5). The faceted search view consists of three key elements: 1) the facet menu on the left, 2) the results view on the right, and 3) the different visualization tabs on top of the results view.

The facet menu includes all the available facets that can be used to filter the data in that particular perspective. The Sampo-UI framework offers various types of facets for different types of data. The BookSampo Portal utilizes three different facets types:

1. **Checkbox facet.** A facet for filtering results by selecting one or multiple checkboxes for
wanted property value entities. Results are automatically updated when a checkbox is checked. Selecting multiple checkboxes works in a disjunctive way: Selecting both the genres *romance* and *thriller* for the genre facet would return works that belong to either (or both) genre. If the property values have a hierarchical structure, e.g., yearly literary awards (*Finlandia Prize 2022*) and the award series they belong to (*Finlandia Prizes*), the facet can be configured to show the entities hierarchically as well.

2. **Integer range facet.** A facet for filtering results by limiting the integer range a property’s value should be in, e.g. searching for works that have a *page count* in the range of 200–300. The facet is applied by pressing the ’apply’ button.

3. **Text facet.** A facet for searching for results based on text string, e.g. searching for works by their *names*. The facet is applied after the user presses enter.

The results are shown in a table format on the right side of the screen. Each of the rows represents one entity. The different columns represent the different properties and property values these entities have. Column values that are underlined denote links to more information about that particular entity. By default the result set includes all entities that match the type of the application perspective with no filters applied. The results are automatically updated when any facets have been applied.

The links in the results table lead to the **instance pages** (shown in Fig. 6) of entities. Instance pages aggregate all the information about that particular object in the same page. This includes the information shown about the entity in the table view as well as possible further information not deemed relevant to be included in the table view. In the BookSampo Portal the general choice was to include information that could be used as facets as columns in the table view and leave the rest of the relevant information to the instance pages.

Similarly to the faceted search view, the instance page view can have multiple tabs for different ways of visualizing the data. These tabs depend on the type of the entity in question. Novels
Figure 6: The instance page of Vuorio Hannu’s novel Miami in the BookSampo UI.

and nonfiction books, for example, have a specific tab for showing detailed information about the different publications of that particular work (shown in Fig. 7) that exist in the data.

Figure 7: The editions list of Hannu Vuorio’s novel Miami; the singular edition of the work is shown.

Visualizing the data with integrated data-analytic tools The various tabs available in the faceted search views and instance pages of application perspectives offer the user different ways of looking at the data as well as analyzing the data. The different visualization types available can be roughly split into three different categories:

1. **Pie/bar charts**. Charts that show the ratio of property values in comparison to each other, e.g., the top genres of novels (shown in Fig. 8). The component behind these visualizations
is created with the ApexCharts\(^7\) library.

2. **Maps.** Charts that show entities on a map based on some location information related to the entity, e.g., settings of novels on a map (shown in Fig. 9). The components behind map visualizations are created with the Leaflet\(^8\) and deck.gl\(^9\) libraries.

3. **Time series.** Charts that show the evolution of entities or some of their properties as a function of time, e.g., the evolution of average page counts throughout years (shown in Fig. 10). The components behind these visualizations are created with the ApexCharts and AmCharts\(^10\) libraries.

![Pie chart visualizing genre](image)

**Figure 8:** The top genres of novels visualized as a pie chart.

The Sampo-UI framework offers some ready-to-use visualization types for all of these categories. To better utilize the potential of the data, some custom visualizations were developed. These visualizations use the libraries already included in the framework, but have custom configuration options, mappers and/or data processing functions to expand on the possible visualization types available in the base Sampo-UI framework. The inspiration behind these components came from the data itself, where no existing visualization type could readily be used to visualize an aspect that could potentially turn out to be interesting.

Two examples of such components are shown in Fig. 11 and Fig. 12. Fig. 11 shows a deck.gl-based component for visualizing the gender ratio of authors, who have written novels with a location as a novel’s setting. The circles on the map represent different locations, where the size of the circle is determined by the number of novels with that location as their setting and the color of the circle represents the gender ratio for that location. Red circles indicate a

---

\(^7\)https://apexcharts.com/
\(^8\)https://leafletjs.com/
\(^9\)https://deck.gl/
\(^10\)https://www.amcharts.com/
strong female author majority and blue circles a strong male author majority, with the purplish shades in between indicating a more even split between female and male authors. In Fig. 11, for example, interestingly the major locations in the UK have at least a slight female author majority, while Germany on the other hand has more locations with a male author majority. With the faceted search tools the UI offers, the user could easily explore in more detail whether, for example, the top genres for these locations could play a part in the difference.

Fig. 12 is based on an existing ApexCharts time series component configuration, but with new data preprocessing functions to enable it to handle dynamic sets of series. This enables the visualization component to render the top themes and keywords and their evolution throughout the years for publications dynamically based on the applied facets instead of having to
Figure 11: A map showing the gender ratio of the authors whose novels have a place as their setting.

Figure 12: A timeline visualization showing the top themes and keywords of original Finnish novels.

predetermine the visualized themes and keywords. In the bottom visualization for the top 10 keywords, for example, there is an interesting visible spike for two keywords – *kielletty kirjat*¹¹ and *myrkkykaappi¹²* – during the late 1930s to 1940s, when the Winter and Continuation Wars were fought in Finland.

The general idea behind the inclusion of visualizations is that they should visualize the entities being filtered in that particular perspective. For example, in the Novels perspective all the visualizations mainly deal with the novels and their properties. Ideally the set of the

¹¹Finnish for ‘banned books’.
¹²Lit. ‘poison cabinet’. This is a term used for a cabinet in certain libraries that would be used to store controversial/banned books.
visualized entities should be the exact same result set as shown in the table view of the results. The problem, however, arises with properties having varying annotation coverage. This is especially apparent with the map visualizations, where the ratio of the set of visualized entities to the whole result set can become low, e.g., only around 12,000 people out of 62,000 people having an annotated place of birth. The problem is also further exacerbated by issues with the geographical location data in the BookSampo KG, where a single location can have multiple different latitudes and longitudes listed in the data. In these cases those locations are just filtered out of the results to prevent them from skewing the visualization by showing the ‘same location’ and thus instance counts at multiple different locations.

The choice on whether to even include a visualization in the first place was based on how potentially interesting the data itself could be: The settings of novels as well as the places of birth and death could potentially show some interesting patterns and correlations between factors, but on the other hand something like the place of education like won’t offer much due to the extremely low annotation coverage (less than 2,000 people out of 62,000 people) as well as the most common annotations largely being Finnish cities with major universities.

For pie/bar charts this problem with coverage is somewhat easier to mitigate by being able to visually show the ratio of known and unknown values by including a slice with the label ‘Unknown’. This also helps the user more easily assess how well the annotated values might represent the whole data set. For map visualizations and most of the time series visualizations this ratio of known to unknown isn’t as clear unless the user manually counts together the instance counts in the visualizations.

5. Discussion

Related Works Linked Data and ontologies have been used in libraries [17], museums [18, 19, 20], and archives [21, 22]. Using LD is advocated by major library organizations, such as IFLA and OCLC, and several libraries provide their collections as data in this form [23]. LD has been used in building infrastructures, such as ARIADNEplus for archaeology, Linked Art in the U.S., and in local efforts in Italy [24], the U.K. [25], Spain [26], and Finland [27] to list a few examples. Cultural Heritage and DH have become a major application domain for LD technologies [28, 29]. The focus in related research on portal UIs has been usually on (explorative) search and browsing [4]. In contrast, Sampo systems have a strong focus on data analysis and visualizations integrated seamlessly with (faceted) search and browsing.

Evaluation The new BookSampo UI has been available mostly to members of the Semantic Computing Research Group as well as people from the Finnish Public Libraries while the most important issues of the underlying KG are being corrected. The UI has been improved based on the wants and needs of the current users, e.g. by including additional facets the users felt the UI was missing and by implementing new custom visualizations to better visualize aspects of the data, but hasn’t yet been evaluated with outside testers. Based on previous in-use semantic
portals utilizing the Sampo-UI framework, the framework is however suggested to have good usability and scalability for the end user [30, 5, 31].

In terms of the evaluation of the contents of the BookSampo Portal itself, the Sampo-UI framework’s structure enables easily developing future improvements. The Sampo-UI framework offers various ways to extend and customize the portal through things like custom components based on the needs of the users, which was already done during development based on the comments from the members of the research group as well as the people from the public libraries. Therefore possible problems or lacking features could be implemented and/or fixed in the future when the portal is made public.

The BookSampo KG has already shown itself to be interesting from the perspective of DH research [32], which the new UI could potentially make more accessible to both non-academic people as well as researchers not previously familiar with the data and its potential. The new UI offers ways to explore and browse the data without the need for much technical knowledge or skills in query or programming languages like SPARQL and Python that are often used when conducting research with these kinds of data sets.

The ease of use, however, also means less ability for the user to affect the data being visualized. For example, the data on geographical locations has its problems with the excess longitude and latitude values as well as a missing hierarchy system for connecting cities and towns to the countries they are in. Issues like these would greatly benefit from the ability to preprocess the data before visualizing it, which would be available to the user if they were working directly with SPARQL and Python instead.

Another problem arises from the nature of simply how things are annotated. The annotations are done by humans who naturally might have their own ways and preferences of how they annotate things. For example, one person might annotate a book’s setting as 'Helsinki' while another might annotate it as 'Finland'. Both of these choices could be correct on a level, but the final choice is dependant on the annotator. With geographical locations, adding hierarchy and visualizing things on a country level could lead to most accurate visualizations in some cases, but with something like themes and genres, aggregating things to an upper shared concept might not be as easy as stories with similar elements can still be described in different ways.

**Future Work** The new UI offers the users a new and more intuitive way to explore the BookSampo KG compared to the old portal with its limited search functions. With the integrated data-analytic tools the users can also easily analyze the data they are presented without having to learn languages like Python to work with the result data from SPARQL queries. The new UI could potentially serve as an intermediary step in DH research. The underlying BookSampo KG has a lot of potential to be used in literary DH research, but has not widely been used for it yet. With the new UI researchers could easily search, explore, and analyze the underlying data without needing to be familiar with technology like SPARQL or Python that would be needed otherwise. In addition to not needing to be familiar with the technology, the UI could be helpful in just narrowing down the are of interest to be researched. In comparison to trying to scour through the KG itself with its nearly 9 million triples, finding interesting topics or questions is be easier with the new UI, where one can expand on search in an iterative way by adding new restrictions on the results through the facets as you go.

Due to the nature of the Sampo-UI framework, the portal could easily be extended in the future. The BookSampo KG includes data on other types of works, e.g., poems and short story
collections, that could easily be added to the portal as perspectives after assessing the quality of that data and the annotations related to them. The components and visualizations present in the portal could be expanded as well based on the needs and wishes of users and their feedback. The data could also be enriched by linking it to other linked data sources, e.g., BIOGRAPHY Sampo KG [33], which includes data on over 700 people present in the BOOKSAMPO KG.

The new portal presented in this paper is planned to be opened for public use in autumn 2023. Before this can be done, the data issues need are being addressed by the Public Libraries of Finland. We also plan to publish (most of) the data for open use with the CC BY 4.0 license at the Linked Data Finland platform\textsuperscript{17}.

Acknowledgements Thanks to Matti Sarmela, Kaisa Hypén, and Tuomas Aitonurmi for their collaborations as well as providing a newer version of the BOOKSAMPO KG to be used in the development of the new user interface. This project was funded by the Aalto University; Computing resources of the CSC – IT Center for Science were used.

References


[26] F. Hernández, L. Rodrigo, J. Contreras, F. Carbone, Building a cultural heritage ontology


Towards a UML-based notation for OWL ontologies

María Poveda-Villalón1,∗, Serge Chávez-Feria1, Sergio Carulli-Pérez1 and Raúl García-Castro1

1Universidad Politécnica de Madrid, Madrid, Spain

Abstract
Ontology conceptualization is a crucial task within the ontology development process. During this activity, developers should generate a conceptual model of the ontology based on the requirements that the ontology should meet. Very often, this activity is carried out by producing diagrams to represent the ontology structure and main elements; to do this, paper, blackboard, or digital drawing tools could be used. From this point on, the generated models drive the implementation of the ontology. Therefore, the closer the diagrams are to the OWL constructs, the easier the ontology developer would be to generate the ontology code. However, up to now, there is no standard notation for OWL that reduces the freedom of modelling between custom diagramming styles and the OWL code. For this reason, in this paper we introduce the Chowlk notation for the OWL language based on UML. This notation is being adopted in several projects including ontologies developed by standardization bodies.

Keywords
Ontology conceptualization, Ontology notation, Knowledge representation

1. Introduction

According to the NeOn methodology “Ontology Conceptualization” refers to: the activity of organizing and structuring the information (data, knowledge, etc.), obtained during the acquisition process, into meaningful models at the knowledge level according to the ontology specification document. This activity is independent of the way in which the implementation of the ontology will be carried out. (definition taken literally from [1]). For carrying out this activity, even though it is not the only solution, ontology developers often use graphical representations as it is more convenient to provide an overall idea of the concepts and relations involved in the model, and it is a powerful tool to communicate the model to potential users or domain experts.

The NeOn definition notes that the conceptualization is independent of the ontology implementation. However, it is advisable to use a conceptualization mechanism as close as possible to the ontology implementation language in order to avoid ambiguity, reducing therefore the ontology developers’ effort when translating the conceptualization to the actual code. For this reason, as a result of the fact that there is no standard notation for graphically representing ontologies, in this paper a UML based notation for ontology conceptualization targeting OWL is presented in Section 2. This notation is compared with the existing efforts in Section 3.

VOILA23: Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs

mpoveda@fi.upm.es (M. Poveda-Villalón); serge.chavez.feria@upm.es (S. Chávez-Feria);
sergiomario.carulli.perez@upm.es (S. Carulli-Pérez); rgarcia@fi.upm.es (R. García-Castro)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)
2. The Chowlk Visual notation

The Chowlk visual notation presented in this paper is based on the UML_Ont profile [2]. It should be mentioned that while the original UML_Ont profile utilizes custom stereotypes and dependencies to cover OWL 1, the Chowlk notation binds the stereotypes used in the profile to OWL, RDF(S) constructs and some OWL 2 constructs. Another reason for developing the Chowlk notation is to provide more compact alternatives to represent property characteristics and axioms than the UML_Ont profile. A preliminary version of the notation was presented in [3] as guidelines for ontology documentation. The following sections detail the diagram elements used to represent OWL ontologies metadata, concepts, properties, and individuals.

2.1. Basic Elements

Figure 1 shows the basic elements of the Chowlk visual notation. First of all, to represent the ontology and its elements URIs the notation should allow the definition of namespaces to make more compact representations. In addition, namespaces and prefixes should be declared in the conceptualization in order to avoid ambiguity in the declaration of the elements, for example, to use two concepts from different ontologies but with the same identifier. A specific block is included in the notation for the declaration of these namespaces (Figure 1 (a)).

Another specific block is defined in the notation to allow the declaration of ontology metadata (Figure 1 (b)) by listing the annotation properties needed and their values for the given ontology. Annotation properties should have the prefix and name of the property. The prefixes of the properties should also be defined in the Namespaces declaration block mentioned above. This block is also used to declare owl:imports.

Figure 1 (c) to (h) represent the rest of the basic visual elements used by the Chowlk notation. These main elements constitute the basis of an ontology conceptualization and have one or more OWL counterparts, that is, each element may be translated into different OWL elements depending on how the elements is used in the conceptualization. The possible OWL elements represented are shown in Figure 1 under each shape. For example, classes and individuals can be represented by rectangles (Figure 1 (c)). In this case, if the URI of the element is underlined, it represents an individual, and it would be a class if not underlined, as will be explained later. This shape, the rectangle, could be also used to represent datatype properties.

When referring to RDF/RDFS/OWL constructs URIs, the double angled brackets (« ») are used around the specific RDF/RDFS/OWL URIs included in the diagrams.

![Figure 1: Chowlk visual notation basic elements.](image-url)

More detailed information about the notation is provided at https://chowlk.linkeddata.es/notation.html.
2.2. Classes

Named classes are represented by a rectangle that includes the class URI (Figure 2 (a)). “Subclass of” is represented by an empty arrow (Figure 2 (b.1)) or using the \texttt{rdfs:subClassOf} stereotype within a dashed arrow (Figure 2 (b.2)). Anonymous classes are defined by an empty rectangle (Figure 2 (h.2)) or circles representing \texttt{owl:intersectionOf} or \texttt{owl:unionOf} (Figure 2 (c) and (d)). Equivalence and disjointness between classes are represented by circles as shown in Figure 2 (e) and (f). The circles with the logic operators inside them represent the axioms, meanwhile the classes connected to those circle using arrows are the concepts involved in the relationship. Classes enumerations are defined using the hexagon linking to the involved individuals (Figure 2 (g)).

Classes defined using property restrictions can be represented concisely by including the operators used in the arrows representing the object properties (Figure 2 (h.1), (j), (l)) or before the datatype properties (Figure 2 (i), (k), (m)). As shown in Figure 2, the tags (all) and (some) indicate existential and universal restrictions, respectively. The tag \texttt{(N1..N2)} represents cardinality. If the user wants to specify an exact cardinality, the numbers \textit{N1} and \textit{N2} should be equal. The notation has the “\textit{N}” letter reserved for the cases in which no maximum cardinality is needed. For instance, the tag \texttt{(2..N)} will indicate a minimum cardinality restriction of 2, and a nonexistent maximum cardinality restriction.

As the notation used in Figure 2 (h.1), (j) and (l) for object properties (and (i) (k) and (m) for datatype properties) could be combined with subclass of and equivalent class axioms, it should be defined to which OWL construct it would correspond. The decision was to understand these options as “subclass of” because the logical commitment is less strong than an equivalent class in terms of reasoning and because it is more likely to find that pattern in existing ontologies. In order to allow for having equivalent classes with the universal, existential and cardinality constraints, the tag \texttt{“(eq)”} is added before the constraint, as shown in Figure 2 (n.1), (o) and (p). It should be mentioned that options (h.1) represent the same modelling as (h.2) and the same holds between (n.1) and (n.2).

2.3. Object properties

Object properties are represented by labeled directed arrows where the head of the arrow indicates the possible domain and range of the property. Several variations can be applied to the style of the arrow depending on their individual particularities. Figure 3 options (a.1) to (d.2), provides a summary of the possible combinations. More precisely, if the property domain is the attached class, the dot at the origin of the arrow is filled (Figure 3 (a.2) and (b)), otherwise it is empty (Figure 3 (c) and (d.2)). If the property range is the attached class, the arrow is solid (Figure 3 (a.2) and (c)), otherwise it is dashed (Figure 3 (b) and (d.2)). To simplify the cases where both the domain and the range are defined, the plain arrow without a circle at the origin could be used as a shortcut (Figure 3 (a.1)) and the same could be used for cases where neither the domain nor the range are defined using the dashed arrow (Figure 3 (d.1)). This notation feature is designed to allow the developer to state between which classes the property intended is to be used, even though there is no domain and range declared. This graphical information is very valuable to a potential user when starting reading an ontology as it is not explicitly declared in
Figure 2: Class definitions, axioms and constraints.
The property characteristics are also indicated by special tags in the labels. The tags (F), (S), (IF), (T) indicate if a property is $\text{owl:FunctionalProperty}$ (e), $\text{owl:SymmetricProperty}$ (f), $\text{owl:InverseFunctionalProperty}$ (g) or $\text{owl:TransitiveProperty}$ (h) respectively. One of the advantages of using tags to declare things like property restrictions (Figure 2 (h.1) to (n.2)) and property types is that they do not interfere with other definitions of properties, such as the domain and range, which control the style of the arrow. The inclusion of concise tags avoids the use of additional elements that will clutter the conceptualization.

The notation also provides an alternative representation for properties using the diamond block (see Figure 1). This shape, reused from UML_Ont profile [2], supports several use cases, especially to declare relationships between properties such as: $\text{rdfs:subPropertyOf}$, $\text{owl:inverseOf}$, or $\text{owl:equivalentProperty}$ to avoid cluttering the main diagram. However, this could also be defined using the arrow notation as shown in Figure 3 (i) to (i).

Figure 3: Object property definitions, characteristics and relations.
2.4. Datatype properties

Visually, datatype properties or attributes are shown as complementary rectangles below classes, as shown in Figure 4. A rectangle can contain more than one attribute. As in the case of object properties, the user can modify the style and text inside those rectangles in order to represent whether domain and ranges are defined. A solid border line of the rectangle indicates that the domain of that set of attributes is the class defined above them (Figure 4 (a) and (b)). A dashed line indicates that the attribute is intended to be used for instances of the attached class; however, the domain is not declared (Figure 4 (c) and (d)). The range is indicated by declaring a datatype in the text of the attribute (Figure 4 (a) and (c)).

Restrictions over the attributes are also possible using the same tags explained for the object properties in and shown in Figure 2 (i) (universal), (k) (existential) and (m) (cardinality). In the case of property types, only the functional tag (F) applies to attributes, according to the OWL specification. If a datatype is functional, it is defined using a tag as shown in Figure 4 (e).

The notation also allows representing the datatype relations using the diamond block to define rdfs:subPropertyOf (Figure 4 (f)), owl:equivalentProperty (Figure 4 (g)) and owl:propertyDisjointWith (Figure 4 (h)).

Figure 4: Datatype property definitions, characteristics and relations.
2.5. Individuals

The Chowlk notation uses the same rectangular visual block to represent classes and individuals; however, the individuals have their labels underlined (Figure 5 (a)). Figure 5 shows four options to state the class membership of an individual (b.1 to b.4)). The first option (b.1) where the individual is attached to the class it belongs to is the preferred one, as it is more compact and reduces the number of arrows. If an individual belongs to more than one class, (b.1) and (b.3) are recommended.

![Diagram of individual definitions and use.](image)

The notation also allows for the definition of individuals’ identity axioms. Two equal individuals can be represented by a solid arrow connecting them with the \texttt{owl:sameAs} construct (Figure 5 (c)) and to state that two individuals cannot represent the same entity the \texttt{owl:differentFrom} construct can be declared (Figure 5 (d)). To indicate that a set of individuals are all different, the hexagonal visual block shown in Figure 5 (e) could be used. In this case, the hexagonal label contains the axiom \texttt{owl:AllDifferent} and is connected to all individuals involved in the axiom using dashed arrows.

It is also possible to instantiate the property values as shown in Figure 5. In the case of object properties, the individuals are connected by arrows stating the relation in the label (f.1). In the
case of datatype properties, the same artifact is used, but now the destination of the arrow is connected to a datatype value (f.2) and (f.3).

3. Related work

During the last decades, ontology visualization and graphical edition have attracted researchers attention. For example, the systematic literature review provided by [4] in which the relation between UML and ontology based formalisms is investigated through the analysis of 66 works. In addition, a comprehensive analysis of visualization methods and tools and their capabilities is provided by [5]. However, in this paper we focus on the notation aspect independently of the technology transforming the conceptualization into code.

In this sense, we can compare our work to existing notations which could be broadly classified in directed graphs oriented or UML alike. One of the well-known of the former type is WebVOWL [6]. In this case, the type of notation used makes it more complex to state a number of object properties between the same two classes. It also generates less compact diagrams as the datatype properties are also arcs to new nodes (therefore, more nodes are needed than in UML-alike visualizations). Finally, the hierarchies cannot be placed in a tree-oriented way as the visualization approach follows a circular shape. A closer notation to UML is the Graffoo specification. In this case, the issue with more cluttered diagrams because datatype properties are also represented with nodes remains. In addition, the class constraints are also represented by an additional box, in contrast to reusing the object properties arrows as proposed by the Chowlk notation, generating cleaner diagrams.

Both WebVOWL and Graffoo use colours in the shapes to convey information about the model, which can be lost, for example, when printing in black and white or for people with different colour perception. In the case of the Chowlk notation, the colour of the classes are meaningless from the conceptualization point of view but it is advisable to use different colours for different namespaces so that the user can easily identify in which ontology is every concept defined. See, for example, the SAREF4CY ontology.\(^2\)

Another notation following UML is OWLGrEd [7]. However, from its notation description\(^3\) it is not clear which OWL features are supported, for example imports, class equivalence, property equivalences, cardinality constraints, etc.

4. Conclusions and future work

This paper has presented foremost characteristics of the Chowlk notation and major advantages over existing ones. Also, the main rationales for some notation characteristics, such as domain and ranges options and the use of tags for property characteristics and class constraints, have been explained. This notation is provided within the Chowlk framework and two diagrams.net libraries are available containing a complete version or a subset of the notation elements.\(^4\)

\(^2\)https://saref.etsi.org/saref4city/
\(^3\)http://owlged.lumii.lv/notation
\(^4\)The lightweight and complete libraries are available at https://chowlk.linkeddata.es/notation.html
The notation has been used in a number of projects including ontologies developed by standardization bodies as, for example, the ETSI SAREF ontologies\(^5\) as it can be seen in the SAREF4AGRI ontology\(^6\) among others. Furthermore, by W3C communities, such as the RML ontologies\(^7\) and the WoT discovery ontology.\(^8\) Other examples are European projects as BIMERR,\(^9\) and COGITO.\(^10\) It is being adopted by projects carried out by organizations not related to the authors of this paper as BIM4EEB\(^11\), CosWOT\(^12\), D2KAB\(^13\), and Mat-o-Lab.\(^14\)

The main future line of work involves the detailed comparison with existing notations in terms of capabilities, clarity, and ability to generate compact diagrams. The notation documentation is planned to be improved overall incorporating more examples. The generation of examples for well-known ontologies is part of the Chowlk roadmap in order to ease their reuse.

Finally, as the notation is used by the Chowlk converter to generate OWL code, it is needed to define notation characteristics to indicate the converter from where to start reading axioms in order to reduce the complexity when processing the notation underlying graph. However, this need is oriented to the technological side of the framework rather than to the visualization capabilities to represent OWL ontologies.

**Acknowledgments**

This work has been supported by COGITO funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 958310 and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with the Universidad Politécnica de Madrid in the Excellence Programme for University Teaching Staff, in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

**References**


\(^{5}\)https://saref.etsi.org/extensions.html
\(^{6}\)https://saref.etsi.org/saref4agri/
\(^{7}\)https://kg-construct.github.io/rml-resources/portal/
\(^{8}\)https://github.com/w3c/wot-discovery/tree/main/context
\(^{9}\)https://bimerr.iot.linkeddata.es/
\(^{10}\)https://cogito.iot.linkeddata.es/
\(^{11}\)https://digitalconstruction.github.io/v/0.5/index.html
\(^{12}\)https://coswot.gitlab.io/
\(^{13}\)http://www.elzeard.co/ontologies/c3po/plant#
\(^{14}\)https://github.com/Mat-O-Lab/MSEO


How to create easily a data analytic semantic portal on top of a SPARQL endpoint: introducing the configurable Sampo-UI framework

Heikki Rantala¹, Annastiina Ahola¹, Esko Ikkala¹ and Eero Hyvönen¹,²

¹Semantic Computing Research Group (SeCo), Aalto University, Finland
https://seco.cs.aalto.fi, firstname.lastname@aalto.fi
²Helsinki Centre for Digital Humanities (HELDIG), University of Helsinki, Finland

Abstract
Sampo-UI is a JavaScript (JS) framework that can be quickly configured to create web applications with powerful search, browsing, and data-analytic visualization tools based on SPARQL² queries to RDF³ triplestores. The user interface (UI) with its components can be created by re-using and configuring a generic UI template and a set of existing UI components whose usage are specified declaratively in JSON configuration files. Using the framework, a first working application with faceted search and various interactive visualization, such as maps and charts, can be created in only a few hours for an existing RDF service with a SPARQL API even with limited programming skills.

Based on the Sampo model [2], a Sampo-UI interface contains a landing page with a set of “application perspectives” to choose from. Each perspective provides the end-user with a faceted semantic search view to filter out individuals of a class (e.g., Person, Place, Artifact, etc.) related to the perspective. After finding a set on individuals of interest, the user can either study and analyze them 1) one by one, 2) analyze sets of individuals together (e.g., statistically, on maps, and graphs), or explore relationships between individuals through various visualizations such as maps, graphs, and charts.

1. Introduction
Sampo-UI [1] is a JavaScript (JS) framework that can be quickly¹ configured to create web applications with powerful search, browsing, and data-analytic visualization tools based on SPARQL² queries to RDF³ triplestores. The user interface (UI) with its components can be created by re-using and configuring a generic UI template and a set of existing UI components whose usage are specified declaratively in JSON configuration files. Using the framework, a first working application with faceted search and various interactive visualization, such as maps and charts, can be created in only a few hours for an existing RDF service with a SPARQL API even with limited programming skills.

To illustrate this, in the video https://vimeo.com/manage/videos/817028609 a simple, but working portal is created in less than an hour.
or on timelines), or 3) explore the data by browsing.

Our earlier article [1] describes the idea, model, and architecture of the initial version of Sampo-UI in detail. In contrast, this paper introduces a new version of Sampo-UI. The biggest change is the new way of configuring the portals using JSON4 files. The main contribution of this article is to show how “Sampo” portals can be configured with JSON files, how the SPARQL queries are formed, and how the various data analyses and visualizations can be created using Sampo-UI. To test and evaluate the new Sampo-UI version, it has been used to create some 15 portals in use in the Sampo series5 [2] of Linked Open Data (LOD) systems that demonstrate practical usability of the framework. The example portal6 of the framework is mainly based on the Mapping Manuscripts Migrations [3] portal. In this paper we use examples from a simple portal which is created in the tutorial available for Sampo-UI. The portal uses the DBpedia [4] endpoint to search writers.

The paper is organized as follows. First, related works are discussed. After this requirements for the Sampo-UI framework are presented and its UI logic is explained. It is then presented how a semantic Sampo portal can be created using the framework and its configuration files. In the final section, contributions of the paper are summarized and discussed and direction for future work is outlined.

2. Related work

The possibilities and benefits of systems utilizing faceted search have long been talked about, especially in the digital library domain [5, 6]. With the rise of linked data, various systems implementing faceted search capabilities on top of RDF data have been developed. The Ontogator [7] browser was an early implementation of a system with a semantic faceted search over RDF data. The OntoViews semantic web portal tool [8] made it possible to create semantic portals with faceted search by utilizing the Ontogator as a search engine. The OntoViews tool led to the development and publication in 2004 of the first Sampo portal, MuseumFinland [9].

The logic of faceted search with SPARQL queries is described extensively in [10]. Multiple faceted search systems and their faceted ranking methods have been surveyed in [11]. Applying faceted search to DBpedia has been implemented before for example in [12]. SPARQL Faceter [13] is a tool for applying faceted search with SPARQL queries fully in the client side of an application7. How SPARQL Faceter has been applied for implementing faceted search and visualizations is presented in [13, 14]. In contrast to SPARQL Faceter and many other faceted search engines, the faceted search capabilities are not the only focus of Sampo-UI: the other focus is on data analytical tools to analyze and visualize the data using a variety options, such as charts, timelines, maps, and network analytics.

There exist multiple faceted search systems that facilitate faceted search of RDF data. For example, SemFacet [15] is a prototype of a system for faceted search of RDF data that can be configured to an extent through a graphic UI. A recent example is Rhizomer [16], which

---

4https://www.json.org/
5For a full list of Sampo systems see: https://seco.cs.aalto.fi/applications/sampo/
6https://sampo-ui.demo.seco.cs.aalto.fi/
7DBpedia writers have been also used to demo the use of SPARQL Faceter. See: https://github.com/SemanticComputing/sparql-faceter-dbpedia-demo.
automatically generates facets for exploration and certain visualisation, such as word clouds, from the data. Rhizomer is already used in certain research projects to explore and understand the research data. In contrast to these kind of systems Sampo-UI is mainly aimed for developers to use as a basis for developing new systems that employ faceted search and visualisations.

There have been also been portals with faceted search functionality in the cultural heritage (CH) domain that integrate data analytical tools for the user. The Nomisma portals\(^8\) in the numismatic domain, for example, provide ways of performing faceted search on numismatic data as well as visualizing it on maps. In the field of archaeology, the ARIADNE portal\(^9\) similarly not only offers the faceted search capabilities but also a few ways of visualizing the data in different formats, such as on a map as well as on a timeline. The ResearchSpace \([17]\) tool of British Museum aims to be a comprehensive platform for conducting CH research. ResearchSpace offers not only tools for exploring and visualising the data, but also tools for updating and adding to the data.

### 3. A framework for creating data-analytic semantic portals

The Sampo-UI framework is based on experiences gathered is developing the so-called Sampo\(^10\) series of LOD services and portals for collaborative CH LOD publishing and research, based on the “Sampo model” \([2]\). This development work has been going on for over twenty years spanning numerous CH projects and over twenty Sampo systems\(^11\) either in use online or in development. The main goal of developing Sampo-UI has been to make it easier for both the users and developers of new applications by trying to standardize the process of creating new applications as well as the way their user interfaces work.

**Requirements for the framework** Based on the Sampo model principles, Sampo-UI was designed and implemented with the following major requirements or goals in mind:

1. Create a consolidated, “standardized” UI logic for semantic portals, based on Semantic Web standards and best practices of W3C. In this way the portals would be easy use for end-users and to implement and maintain for the developers.

2. Create semantically rich functionalities for semantic search and semantic browsing (data exploration), integrated seamlessly with data-analytic and visualization tools. A particular application area of research in our work has been Digital Humanities. \([18]\)

3. The framework should be re-usable on top different external SPARQL endpoints without touching the underlying data or LOD service. In this way the tool could be used, e.g., on top of LOD services on the Web, such DBpedia, Wikidata, Getty thesauri, and the CH LOD services of the Linked Data Finland platform\(^12\) hosting the KGs of different Sampo systems.

---

\(^8\)https://nomisma.org/about/

\(^9\)https://portal.ariadne-infrastructure.eu/

\(^10\)The name ‘Sampo’ comes from the Finnish epic Kalevala: it refers to a mythical artefact and is a metaphor of ancient technology that brings wealth and fortune to its owner.

\(^11\)Sampo portals homepage: https://seco.cs.aalto.fi/applications/sampo

\(^12\)https://ldf.fi
4. The framework should be *adaptable* to conform to different metadata data models [19] and ontologies used in knowledge graphs, such as Dublin Core, CIDOC CRM, etc.

5. The framework should be *extendable* with new data-analytic and visualization tools needed in different applications and domains.

**Figure 1:** The default navigational page structure of a portal based on Sampo-UI.

**Generic UI logic of Sampo-UI** Fig. 1 illustrates the navigational structure of using a Sampo-UI-based portal. The user first lands on the *landing page* with several *application perspectives* to the data. The perspectives are based on classes of the underlying KG, such as Artefacts, Persons, Places, etc. The usage cycle of each perspective can be divided into two steps: 1) filter and 2) analyze. The user first filters the data by using the faceted semantic search [20] tools provided by the portal. The results as well as the facet options are updated after each selection of a facet, making it possible for the user to precisely filter the end-result entities by different aspects, e.g., filtering by birth country as shown in Fig. 2. After filtering the data to the wanted subset, the user can analyze the results set, i.e., a set of instances of the class corresponding to the application perspective, with integrated data-analytic tools available as tabs on the application perspective page. An example of a visualisation can be seen in Fig. 2, where the number of publication per year of writers (based on the perspective) in DBpedia KG who have been born in United States (based on the faceted selections). It is easy to see that the number of publications peaks in 2008, and then declines. This could then be compared to selections, such as writers from other countries or from different genres.

It is also possible to select a particular instance of the result set for a closer look: each instance has an instance page that provides aggregated information about the individual with internal
and external links for further information to browse. In addition, instance pages also may have a set of tabs that provide contextualized visualisations of the individuals. This filter-analyse two-step usage cycle allows an iterative approach to exploring the data by being able to organically find potentially interesting subsets in the data without having to already be somewhat familiar with it.

Figure 2: Analyzing writers in DBpedia KG using Sampo-UI: “United States” has been selected on the “Birth country” facet, and the “publication year” line chart option has been selected to visualise the result set as a timeline.

4. Creating a portal using Sampo-UI

This section presents the configuration of a new Sampo-UI portal built to work on top of an existing triplestore database accessible through an open SPARQL endpoint, DBpedia. The demo portal presented in this paper also has a live demo available. The portal is intended as a simple tutorial portal, that can be created quickly to learn how to use Sampo-UI.

The application UI with its components are specified declaratively using JSON files in three main directories: 1) configs. JSON files configuring the portal and its perspectives. 2) sparql. SPARQL queries referred to in the configs files. 3) translations. Translations of things like menu items and labels for different locales. When creating a new UI configuration, existing UI components, such as those for the facets and visualization tabs, can be re-used, and the system can also be extended with new components. The Sampo-UI framework consist of a back-end based on Node.js and a JavaScript front–end, based mainly on React and Redux libraries.

Faceted search with SPARQL queries SPARQL is a query language meant to retrieve and change data from RDF databases. Sampo-UI application logic is based largely on creating

---

13 Source code for the portal available at: https://github.com/SemanticComputing/dbpedia-sampo-ui-demo
14 https://www.dbpedia.org/
15 Available at: https://sampo-ui-dbpedi demo.seco.cs.aalto.fi/en/
16 You can see the tutorial at https://seco.cs.aalto.fi/tools/sampo-ui/Sampo-UI-tutorial.pdf. The tutorial portal can be created in around an hour by an experienced user, but a first time learner should reserve more time.
17 https://nodejs.org/en
SPARQL queries based on facet selections made by a user, and then showing the results in appropriate formats to the user. Generally Sampo-UI only requires SPARQL access to the data through an endpoint. The data only needs to follow general principles of RDF, so that most of the features of Sampo-UI framework can be implemented for it. The main exception to this is free text search which does require specific indexing from the endpoint.

Sampo-UI includes two different implementations for faceted search: "client side faceted search" and "server side faceted search". By client side faceted search we refer to an implementation where some set of data is retrieved from an endpoint through a SPARQL query, transmitted to an internet browser of a user, and searched and visualizations are then performed within the browser to that data without any further need for queries to the SPARQL endpoint. This approach has advantages and disadvantages. This kind of faceted search scales well and will generally be quick, however there is a limit for how much information the client browser can process before running out of memory. This means that faceted search can be applied to only a limited number of instances. Either the data need to be relatively small, or there needs to be some sort initial filtering of data before applying faceted search. In contrast the server side faceted search is implemented through SPARQL queries to an endpoint. This means that there are no limits to the number of instances beyond the limitations of the endpoint. However, the way Sampo-UI implements this kind of search means that every facet and all results require separate SPARQL queries that can be demanding for the endpoint.

The SPARQL queries are automatically formed mainly using string forms where parts, for example "<FILTER>" or "<ORDER_BY>" are filled by Sampo-UI based on configuration or facet selections using string replace function. The resulting string is combined with prefix file given separately. The results of the query may then be passed through a mapper function before delivering the results to the front-end as a JavaScript object.

Different types of queries used include: facet values query, result count query, paginated results query, all results query, and results by URI query. In a faceted search perspective generally after every change in facet selections a result count query, a facet values query for every open facet, and some results query is performed. Facet queries get the available selection of values and getting the hit counts for each. Result count query is performed separately, and some kind of results query that can vary depending on what kind of results visualization is selected. Usually as a default the results are shown as a paginated table.

**Configuring application perspectives** The configuration files for the portal in `src/configs` are divided into two levels: the portal level configuration affecting the whole portal (e.g., list of available perspectives and locales, navigation bar configuration, etc.) and configurations specific to single perspectives (e.g., what columns or facets are included in a specific perspective). The perspective configuration files then reference the name of the correct JS file containing the wanted SPARQL queries as well as the appropriate variable name that holds the specific query to be used in that file.

The JS files with the SPARQL queries contain variables holding various queries to be used for the perspective. The exact format of the queries is dependent on their usage and whether that usage passes them to a template (e.g., facet query template) or not; the queries for getting the properties to be shown as results should just include the inside block of the SPARQL `SELECT` queries while queries for visualizations should have the whole query. The developer can define all the prefixes they wish to use in the queries and the query structure itself is going to be
similar to how one would normally write queries for that particular endpoint. In order for the
queries in Sampo-UI to work, the data in the SPARQL endpoint needs to just follow general
principles of RDF, with the exception of the free text search functionality, which requires a
specific indexing configuration from the endpoint.

The key difference in the queries is how the wanted data is captured in variables. The variable
name itself should match the names defined in the configuration files, but the type of the data
matters as well. Literal values can be just captured to the variable of same name, but other
types, e.g., objects, should be captured as objects. This is done by capturing both the ID and
preferred label of the object under the variable name, e.g., ?genre__id for the ID of a genre
and ?genre__prefLabel for the preferred label. This way the data is handled as an object by
the Sampo-UI, which facilitates the developer to, for example, add links to the column values
while still showing the preferred labels.

Fig. 3 illustrates the structure of the faceted search view in the demo portal using
the DBpedia data. The perspective is configured for browsing and searching the writers
(dbo:Writer) present in the DBpedia KG with information on their names, the genre(s)
they have written (dbo:genre), their occupations (dbo:occupation) as well as their almae
matres (dbo:almaMater). The property path can also be longer as in the case of birth countries
(dbo:birthPlace/dbo:country).

Each row in the table represents a singular writer with the values in different columns listing
that particular information for the specific writer. Clicking on the name of any of the writers
would open up that particular writer’s own instance page showing the information about that
writer as table with each row representing a property and its value.

The genres, occupations, almae matres and birth countries are available as facets to be used
for filtering the writers. The facets utilize Sampo-UI framework’s checkbox facet type, where
each of the facet values for a property is shown in a list with checkboxes next to the label.

18The prefix 'dbo' is used for http://dbpedia.org/ontology/
Selecting a checkbox automatically updates the results and the available facet values in the other facets. Selecting multiple checkboxes inside a single facet updates the results to include all results that match at least one of the facet values, while selecting checkboxes from different facets in a conjunctive manner. For example, selecting the Science fiction and Fantasy facet values for the genre and the Novelist facet value for the occupation would filter the results to include only writers who are novelists by occupation and have written at least one of the selected genres, science fiction or fantasy works.

The values inside the facets can be sorted in two ways: 1) in an alphabetical order or 2) based on how often the value occurs in the data. In addition to just the values present in the data, missing values can be aggregated under the label 'Unknown'. The instance counts are listed after the label for the facet value inside brackets and are updated accordingly if the user makes selections in other facets. The users can use the instance counts in the facet to already get an idea of the nature of the data (e.g., most popular genres or occupations for writers) they are browsing. The inclusion of the 'Unknown' category additionally helps the users to gauge the reliability of the data: High 'Unknown' values indicates low annotation rate for this particular property, which means that the top non-'Unknown' values might now represent the data set as a whole. Sampo-UI framework also includes the option to add a pie and/or bar chart button to the facet to automatically generate a pie/bar chart based on the values of that particular facet for visualizing the distribution of relative hit counts. Below is an example of JSON configuration of the 'genre' facet. You can see, for example, that the 'predicate', meaning the property path from the instance of the perspectives class to the facet value, is given as 'dbo:genre', and the 'pieChartButton' option has the value 'true'.

```
"genre": {
  "containerClass": "ten",
  "facetType": "list",
  "filterType": "uriFilter",
  "facetLabelPredicate": "rdfs:label",
  "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
  "predicate": "dbo:genre",
  "searchField": true,
  "sortButton": true,
  "sortBy": "instanceCount",
  "sortByPredicate": "dbo:genre/rdfs:label",
  "sortDirection": "desc",
  "pieChartButton": true
},
```

Configuring visualisations For the results view itself, in addition to being able to see the results as a table, there is another tab for visualizing the distributions in the data as pie or bar charts. For example Fig. 4 shows a pie chart visualisation of birth countries of the writers with science fiction genre selected from facets. It is easy to see that the United States is the birth country in slightly more than half of the cases where the data is available, with UK coming second. This can be easily compared to case where there are no facet selections, and while USA is still number one the proportion is much smaller. On the other hand, it is important to note that in most cases this data is not available in DBpedia, as can be easily seen by looking at the 'Unknown' value in the facets. These pie and bar charts function similarly to the ones that can be included in facets, but offer more freedom to the developer in terms of configuration by having configuration options available as well as using custom queries for generating the results in comparison to the query templates used by the facet menu ones.
Figure 4: A pie chart visualisation of the distribution of birth countries of writers of science fiction genre. USA is the birth country in more than half of the cases where the data is available.

The data for the visualisation is formed through three different steps: 1) the SPARQL query for fetching the relevant results, 2) the mapper that maps the results to only include the relevant variables as well as possibly containing some needed preprocessing for the data and 3) the functions processing the data to the correct format for the visualization library as well taking possible configuration options into account. The template for the visualisation in Fig. 4 is the following:

```
SELECT ?category ?prefLabel (COUNT(DISTINCT ?writer) as ?instanceCount)
WHERE {
  FILTER {
    ?writer a dbo:Writer ;
    FILTER(LANG(?prefLabel) = 'en')
  }
}
GROUP BY ?category ?prefLabel
ORDER BY DESC(?instanceCount)
```

The `<FILTER>` is used for adding the possible filters made with the facets to the query. Because 'science fiction' is selected from the 'genre' facet, and 'writer' is set as the filter target in the configuration file, Sampo-UI will replace the `<FILTER>` in the query with the following lines:

```
VALUES ?genreFilter { <http://dbpedia.org/resource/Science_fiction> }
```

The system also adds prefixes before executing the final query\(^{19}\). If the user adds more filters using the facets, the visualisation is updated accordingly so that the visualised data is the same as the results listed in the table. The results from this query are then mapped with a pie chart

\(^{19}\)You can test the query on Yasgui: https://api.triplydb.com/s/aUHk_t9Hm.
mapping function that takes the values for category, prefLabel and instanceCount from the query results. This particular visualization doesn’t require any additional processing in the mapping process and the results are passed in this form to the relevant data processing functions and the final visualisation component in the front end. In this case the visualisation is created using the ApexCharts\textsuperscript{20} library.

5. Discussion and future work

The Sampo-UI framework is based on years of experiences in creating semantic portals based on RDF data for mainly CH domains, often by people who can be, for example, undergraduate research assistants with basic understanding of programming, but with limited professional skills. These experiences showed that in practice it was usually most convenient to take the code of an existing earlier portal as a starting point, instead of creating a completely new application that would just use some libraries for implementing the search and visualisations. In Sampo-UI this idea is embraced. The framework is essentially a working example application that is meant to be changed to suit the needs of the specific use cases. This means that new applications can be created quickly and with very little knowledge of web development. New visualisations and other improvements can be implemented to the Sampo-UI repository, and then pulled to applications created with the framework. The negative side of this is that the Sampo-UI can be a more complicated tool than many use cases would need, and can include often unnecessary dependencies.

After working with humanities researchers in multiple projects, it is striking how much faceted search and simple visualisation can give new insights to the data even to researchers that are intimately familiar with it. For example, many issues with weird or missing values will be immediately apparent by looking at the facets. However, portals created with Sampo-UI, or similar application, should not be expected to fully replace quantitative analysis. The visualisations are usually limited in some ways, and the exact process of creating the visualisation may not be completely to the researcher when using these kinds of web applications. Their best use is to give insights to the data, by allowing “playing” with the data, and quick testing of hypothesis.

The new JSON configuration file-based version of Sampo-UI was created to make Sampo-UI easier and quicker to use. In this form it is possible to create and customise the portal with very limited understanding of web application development. However, it is still necessary to understand the underlying data and SPARQL queries. Also, customising the application will require some knowledge of programming. This can be challenging for many researchers who might benefit of faceted search analysis of their data. For such uses tools like Rhizomer that do not require any programming skills can be more appropriate.

The approach of Sampo-UI has some limitations. It is created to search instances of classes. For example searching instances of every subclass of a certain class would not currently be easy to implement. Also, faceted search through SPARQL queries can be resource intensive, mostly because of calculating the hit counts in facets. The performance of Sampo-UI queries is related to, at least, the number of instances in the data, and the length of the property paths.

\textsuperscript{20}https://apexcharts.com/
Huge numbers of instances or long property paths between instances and facet values can slow the performance considerably. Sampo-UI is mainly tested on current versions of Apache Jena Fuseki\textsuperscript{21}. Therefore it is possible that some functionalities might not work with other triple stores without alterations. For example, currently to use other than Lucene text search of Fuseki the basic SPARQL queries need to be changed, and this can’t be done through configuration alone. Sampo-UI also does not currently support federated queries in faceted search.

One important reason for using JSON as the basis of configuration is that JSON files are easy to generate with programming languages such as Python or Java. This is intended to make it easy to generate the configurations automatically. We are working on creating tools based on Sampo-UI that can generate portals with RDF configuration or straight from tabular files. We are also working on creating tool based on Sampo-UI that would (semi)automatically analyse parts of the data. In the most simple case it would show the user the missing values in the data, but in a more advanced case it might be able to highlight interesting patterns in the data.

Acknowledgments

The work of writing this paper is partly supported by the Semantic Parliament (ParliamentSampo) project of the Academy of Finland, the EU project InTaVia: In/Tangible European Heritage\textsuperscript{22}, the Jenny and Antti Wihuri Foundation, and Finnish Cultural Foundation. CSC – IT Center for Science has provided computational resources.

References


\textsuperscript{21}https://jena.apache.org/documentation/fuseki2/
\textsuperscript{22}https://intavia.eu/


Short papers
Integrating Sparklis and ViziQuer for Enhanced SPARQL Querying and Visualization

Uldis Bojārs, Jūlija Ovčiņņi, Lelde Lāce, Artūrs Sproģis, Mikus Grasmanis and Kārlis Ķerāns

Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia

Abstract
This paper introduces a multimodal SPARQL query system that combines the capabilities of Sparklis and ViziQuer, two powerful tools for SPARQL query building and visualization. Sparklis offers a faceted user interface for constructing SPARQL queries, while ViziQuer provides a rich visual interface for constructing and visualizing SPARQL queries. By integrating the two applications, the system facilitates the automatic visualization of SPARQL queries constructed in Sparklis within the ViziQuer visual environment.

Keywords
RDF, SPARQL, query visualization, Sparklis, ViziQuer

1. Introduction

SPARQL queries can be difficult to write by non-technical users [1]. There are various SPARQL query building assistants that can help with this task, including tools using form-based interfaces (e.g. PepeSearch [2] and WYSIWYQ [3]), controlled natural language snippets (e.g. Sparklis [4]), and visual diagrams (cf. e.g. [1,5,6,7]).

Each of the notations have their strengths and weaknesses, especially when it comes to the design of rich SPARQL queries (involving, e.g., aggregation and subqueries). For instance, the Sparklis notation allows for rich query composition using the natural language snippets that can be expected to be suitable for a less technical end-user, while the created query formulations may still appear difficult to understand by some end-users. On the other hand, ViziQuer may provide a structural overview and further refinement of the query.

Both Sparklis and ViziQuer provide means for rich SPARQL query definition including complex expressions, grouping and aggregation. The ViziQuer functionality for visualizing SPARQL queries [8] opens the way for combining the strengths of both query building methods into a multi-modal query creation environment, where the query can be initially built in Sparklis and then translated into ViziQuer (from the SPARQL query form that is provided by Sparklis).

This paper presents a prototype of such a multi-modal system that integrates Sparklis and ViziQuer tools and allows users to visualize SPARQL queries created in Sparklis. The integrated system combines the benefits of the two tools: users can use the faceted UI and controlled natural language approach of Sparklis, and can visualize and refine these SPARQL queries in the ViziQuer visual query environment. These visualizations are based on the UML-like notation implemented in ViziQuer and ViziQuer’s query visualization functionality [7,8]. Apart from the particular integration of two SPARQL query composition assistants, SPARKLIS and ViziQuer, this paper establishes the concept of integration of several such assistants into a single multi-modal environment.
The rest of the paper consists of main information about Sparklis (Section 2) and ViziQuer (Section 3), followed by a description of the integration of these tools (Section 4) and examples of SPARQL queries in both notations (Section 5). The paper is completed by a summary of related work (Section 6), and conclusions and future work (Section 7).

2. Sparklis

Sparklis is an open source SPARQL query builder tool that uses controlled natural language and a faceted search user interface, and allows people to explore and query SPARQL endpoints without knowledge of SPARQL or the vocabulary used by a particular SPARQL endpoint [4]. It is a web application that runs entirely in the browser.

Sparklis covers a large subset of SPARQL 1.1 SELECT queries: basic graph patterns (BGP) including cycles, UNION, OPTIONAL, NOT EXISTS, FILTER, BIND, complex expressions, aggregations, GROUP BY and ORDER BY. Its configuration panel offers options to adapt to different endpoints. Sparklis also includes the YASGUI editor to let advanced users access and modify the SPARQL translation of the query. We extend the Sparklis functionality by letting users also access and modify the visual representation of the SPARQL query.

Figure 1: Sparklis query example

The query builder functionality lets Sparklis users incrementally build complex queries by combining elementary queries. Elementary queries can be a class (e.g. "a film"), a property (e.g. "that has a director"), an RDF node (e.g. "Tim Burton"), a reference to another node (e.g. "the film"), or an operator (e.g. "average"). Sparklis queries are verbalized in controlled natural language, hiding the SPARQL queries generated by this tool from the user. Figure 1 shows an example of a Sparklis query for films by Tim Burton that star somebody born after 1980. Additional query examples can be found on the Sparklis website 4.

3. ViziQuer

ViziQuer is an open source UML-style visual SPARQL query tool that allows users to define SPARQL queries visually using the ViziQuer visual notation. The ViziQuer notation and environment (cf. [7,9]) provides visual means for rich query definition, involving BGPs, value filters, optional and negated constructs, as well as unions, aggregation, grouping and subqueries,

---

3 https://github.com/sebferre/sparklis
4 http://www.irisa.fr/LIS/ferre/sparklis/examples.html
5 https://viziquer.lumii.lv/
coming close to the full SPARQL 1.1 SELECT query visualization [8]. ViziQuer source code is available on Github⁶.

ViziQuer has been extended with a query visualization functionality that allows users to transform SPARQL queries into their visual representation. The visualization of SPARQL SELECT queries 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, with important optimizations for representing the variable/resource classes in the dedicated class name compartments and single-use SPARQL triple objects within node attribute fields, so obtaining a compact query presentation [7].

The visualization of a SPARQL query could originally be achieved by copying the SPARQL query text into ViziQuer’s SPARQL query pane and activating a "Visualize SPARQL" context menu item. Before doing this, users also needed to log in to the ViziQuer tool. In order to perform automatic integration of ViziQuer with external tools such as Sparklis, ViziQuer was extended in order to allow anyone to work with SPARQL queries (incl. visualizing queries) without logging in.

4. Implementation

The integration of Sparklis and ViziQuer was realized by creating a Sparklis plugin (viziquer.js) that retrieves the SPARQL query and associated information from Sparklis and sends it to the ViziQuer API. Sparklis source code was augmented in order to add the ViziQuer tab to the Sparklis UI. The user interface allows ViziQuer to be launched either in the same Sparklis screen (button "Show here") or in a new browser tab (button "Show in a new tab"). The modified version of Sparklis and its ViziQuer plugin are available on Github⁷.

![Figure 2: ViziQuer section in the Sparklis query builder](https://github.com/LUMII-Syslab/viziquer/tree/development)

⁶ https://github.com/LUMII-Syslab/viziquer/tree/development
⁷ https://github.com/LUMII-Syslab/sparklis
The required ViziQuer API call can be invoked as an HTTP POST request to the API endpoint provided by the public ViziQuer service. Alternatively, users can launch their own instances of ViziQuer. The parameters of this API call include the type of the query (SPARQL), the SPARQL endpoint URI and the SPARQL query to be visualized along with a flag that indicates if the visualization action should be started automatically (the other alternative is to just send the SPARQL query to ViziQuer and let users launch the visualization action manually). Figure 3 shows the parameters passed to ViziQuer’s JSON API call.

Figure 3: Parameters for the ViziQuer API call

In a similar way, the ViziQuer tool can be integrated into other querying environments, e.g., into a YASGUI based frontend of a SPARQL endpoint.

Query visualization in ViziQuer works best if ViziQuer is "aware" of the data schema of the given SPARQL endpoint but it is also possible to visualize SPARQL queries for endpoints for which ViziQuer does not have a data schema available.

5. Results

Through the integration of Sparklis and ViziQuer, users can visualize SPARQL queries constructed in Sparklis. This section provides some examples of queries expressed in the controlled natural language of Sparklis [4] along with the corresponding SPARQL queries and their visualization in ViziQuer.

UML-style visual queries in ViziQuer notation consist of nodes describing variables or resources where each node can have a possible class name and attribute specification. One of the nodes is marked as the main node of a query (orange round rectangle). The edges that connect the nodes correspond to links among the query variables or resources. Furthermore, condition/filter fields, as well as aggregation and query nesting links can be used in query construction [7,9].

ViziQuer allows users to edit and fine-tune the visual representation of SPARQL queries. We used this functionality to manually tune the presentation of visualizations shown in this section.

Figure 4 shows a simple query for information about biomolecule class instances from DBPedia in Sparklis and ViziQuer representation. The following SPARQL query is represented by Sparklis and ViziQuer notations in this example:

```sparql
SELECT DISTINCT ?Biomolecule_1 ?label_103 ?name_140
WHERE {
  ?Biomolecule_1 a dbo:Biomolecule .
  ?Biomolecule_1 foaf:name ?name_140 .
}
LIMIT 200
```

---

8 https://viziquer.app/api/public-diagram
We observe that the ViziQuer representation of the query matches the structure of the original Sparklis query, explicating at the same time some of the assumptions regarding property and variable names and the query limit that are left implicit in the original Sparklis notation.

Figure 5 illustrates a query that uses aggregation to calculate the number of languages spoken in Colombia.

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT (COUNT(DISTINCT ?Language_1) AS ?number_of_122)
WHERE { ?Language_1 a dbo:Language .
    ?Language_1 dbo:spokenIn dbr:Colombia .}
LIMIT 200
```

Figure 4: SPARQL query for biomolecule information in Sparklis and in ViziQuer

Figure 5: SPARQL query using aggregation in Sparklis and in ViziQuer

The visual query representation distinguishes the nodes corresponding to the language and the country. The full visual appearance of the query is somewhat overburdened by the explicit variable names (Language_1 and number_of_122) introduced by the Sparklis tool in the generated SPARQL query:
Figure 6 illustrates a more complex query that uses aggregation to calculate the number of books written by poets and returns the results ordered in decreasing order of aggregated values. It shows the original Sparklis representation and four options how the corresponding query can be represented in ViziQuer. Option (a) is auto-generated from the SPARQL query produced by Sparklis, while option (b) is a cleaned-up form of it. The options (c) and (d) were created manually by placing the Person class at the main node of the query. Option (c) uses the joined classes construction, while option (d) uses a subquery to calculate the number of books authored by a person (option (d) would translate to a different SPARQL query form).

It might be an interesting future work to understand which of the query representations (b), (c) or (d) would be the easiest-to-understand by various groups of potential end users. This ease of perception may depend on users mastering the concepts of aggregation with grouping (option (b)), instance aliases (option (c)) and subqueries (option (d)). Each of these representations provides an explicit query structure of nodes linked by properties that was not present in the initial Sparklis formulation. Given the obtained understanding, further work would be to implement an automated creation of the desired visual query form.

The following SPARQL query corresponds to the Sparklis and ViziQuer query representation in Figure 6 (a):

```sparql
WHERE { ?Person_1 a dbo:Person .
   ?Person_1 dbo:occupation dbr:Poet .
} GROUP BY ?Person_1 ORDER BY DESC(?number_of_165) LIMIT 200
```
6. Related Work

The visual presentation of information can facilitate its perception. Facet-based tools such as PepeSearch [2] and WYSIWYQ [3] aim to make it easier to create SPARQL queries by using data forms and facets. Visual query composition tools allow users to define the query visually and can be classified into tools that display attribute values in separate graph nodes and tools that use a UML-style notation that offers us a more compact query representation. Examples of the former group are QueryVOWL [6], RDF Explorer [10] and GRUFF [11] which support just the simplest forms of conjunctive SPARQL queries. FedViz [13] lets users visually select query classes and properties, yet it does not provide a full visual representation of the query. FedViz and SPARQLGraph [12] differ from other tools in that they allow users to build federated queries. In the UML-style group, Optique VQS [1] and LinDA [5] also support outer-level aggregation. 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 more limited expression language and expressions are not shown explicitly in the Optique VQS visual query presentation.

Compared to these tools, the ViziQuer notation and environment (cf. [7,8,9]) provides visual means for rich query definition, including basic graph patterns, 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 [8]. Sparklis, while using a different approach – controlled natural language and faceted exploration – also supports a large subset of SPARQL 1.1 SELECT queries (OPTIONAL, NOT EXISTS, FILTER, BIND, etc.) [4].

The integration of Sparklis and ViziQuer was made possible by the extension mechanism in Sparklis and the ViziQuer visualization API. While there are other tools available for defining SPARQL queries, after exploring related work we did not find other instances of integrating SPARQL query tools in the way described in this paper.

7. Conclusions and Future Work

In this paper we presented the integration of two powerful tools for defining SPARQL queries – Sparklis and ViziQuer – resulting in a multi-modal system that allows users to define SPARQL queries in Sparklis and visualize and refine them in ViziQuer. We described the implementation details of this integration and demonstrated it with SPARQL queries in their Sparklis and ViziQuer representation. The obtained results show that the visual query representation that is auto-generated from the technical SPARQL query requires some cleaning up to improve its usefulness for the potential end-users. Implementation of such cleaning is left to future work. It may also be worthwhile to consider making use of higher-level Sparklis concepts in the visual query generation. While it is essential for the ViziQuer tool to maintain the principle of generating the visual queries from their SPARQL encoding, incorporating some form of annotations into this encoding could let us preserve the principal query generation architecture and still reach the necessary effects of user-friendliness.

The core of the Sparklis and ViziQuer integration described in this paper is the ability of the ViziQuer tool to create a visual representation of a given SPARQL query. This feature also allows integrating the SPARQL query visualization functionality in other contexts where SPARQL queries are available. The technical solution of invoking ViziQuer without logging in, described in Section 3, can be used to support such visualizations. It could also be useful in simpler use cases where a visual query environment can be seamlessly offered to the users for visually composing queries over a given SPARQL endpoint.

Other potential areas for future exploration are the integration of ViziQuer and Sparklis in the opposite direction (from ViziQuer visual notation to Sparklis queries), which would require a method for reverse translation of SPARQL queries into their Sparklis representation, as well as the integration of the ViziQuer SPARQL visualization functionality with other Semantic Web tools.
Acknowledgements

This work has been partially supported by a Latvian Science Council Grant lzp-2021/1-0389 “Visual Queries in Distributed Knowledge Graphs”.

References


Towards A Knowledge Graph-based Exploratory Search for Privacy Engineering

Guntur Budi Herwanto¹,²,³, Fajar J. Ekaputra⁴,³, Florina Piroi³ and Marta Sabou⁴

¹Faculty of Computer Science, University of Vienna
²Department of Computer Science and Electronics, Universitas Gadjah Mada
³Institute of Information Systems Engineering, Faculty of Informatics, TU Wien
⁴Institute of Data, Process, and Knowledge Management, WU Wien

Abstract
The concept of privacy-by-design has gained considerable attention in the wake of legal requirements that software systems should fulfill. The interconnected knowledge on privacy concepts, legal requirements, and software development artifacts required to implement this concept can be a major burden for organizations. To address this challenge, we propose a knowledge graph-based Exploratory Search system to help organizations complete privacy engineering tasks. We identify major requirements of such systems and develop an Exploratory Search prototype built on privacy engineering knowledge that integrates different information sources. While still preliminary, this work serves as a foundation for future research on integrating privacy knowledge into software development and demonstrates the potential of Exploratory Search Systems to support the cognitive process in privacy tasks such as privacy threat identification.

Keywords
knowledge graph, exploratory search, privacy engineering, privacy requirement

1. Introduction
The concept of “privacy by design” has come to the spotlight following the implementation of the General Data Protection Regulation (GDPR¹). This concept mandates that software developers embed privacy protection directly into their applications from the start, rather than as an afterthought [1]. However, compliance with these privacy measures can present substantial challenges for software developers [2]. Privacy Engineering has recently emerged as a research area focusing on tackling these challenges [3].

Given the complexity of these various areas of knowledge, Privacy Engineering stakeholders, such as privacy engineers and software developers are often intimidated by the amount of knowledge required to follow privacy engineering principles [4]. They have to rely on various heterogeneous sources to ensure that their work covers the most relevant aspects of privacy requirements. Therefore, it is essential to support them with methods and tools for (i) integration of data from heterogeneous sources, and (ii) intuitive exploration of knowledge.

¹General Data Protection Regulation (GDPR), https://gdpr-info.eu/
The heterogeneous sources range from privacy frameworks, software artifacts, threat intelligence, privacy design patterns to region-specific regulations such as the GDPR and CCPA [5]. Utilizing semantic web technologies, especially ontologies, addresses this by offering a unified knowledge representation, seamlessly interlinking related concepts across diverse datasets. This interconnected framework ensures holistic privacy engineering, facilitating intuitive queries across software design [6], threats, and legal requirements, and promoting scalability as privacy landscapes evolve. Such technologies transform the intimidating breadth of information into a comprehensible and actionable resource for stakeholders.

While ontology-based Exploratory Search strategies have been successfully implemented in domains such as software engineering [6, 7] and the general domain [8], their application in the field of Privacy Engineering remains underexplored. Such strategies hold significant potential for assisting privacy engineers and software developers. Specifically, they can aid in the identification, connection, and modeling of privacy threats and mitigation strategies, processes which are currently conducted manually [9]. The exploration of this approach within Privacy Engineering could fill an essential gap in the literature and practice.

In this paper, we propose the adaptation of an existing Exploratory Search system method [6] and tool [7] equipped with necessary visualization for the privacy engineering context. For this purpose, we utilized an early version of an ontology that we developed, tailored for the privacy engineering context that covers most of the early stages of privacy-aware software development. This knowledge can be encapsulated and made reusable by such a system, streamlining the integration and implementation of privacy-related elements in design activities and, ultimately, ensuring compliance [5].

**Figure 1:** LINDDUN Privacy Threat Analysis Procedure [10]. The SAs shapes the DFD. The PEs elicits privacy threats and requirements, leading to the selection and development of privacy-enhancing solutions together with the SDs.

### 2. Context and Background

Figure 1 depicts a privacy threat modeling process called LINDDUN [10] consisting of six steps: (i) Data Flow Diagram (DFD) definition, (ii) Mapping privacy threats to DFD elements, (iii) Identification of misuse/problematic case scenarios, (iv) Risk-based prioritization of privacy issues, (v) Elicitation of privacy requirements, and (vi) Selection of privacy-enhancing solutions.
We selected LINDDUN as our model for privacy requirements engineering due to its extensive application and widespread acceptance [11]. Threat modeling involves the collaboration of numerous stakeholders. We outline three primary participants in the privacy engineering procedure: (i) Software Architects (SA), who focuses on designing the general architecture of the software, e.g., DFD definition (cf. Step 1 in Figure 1), (ii) Privacy Engineer (PE), who is responsible for identify privacy requirements and propose mitigation strategies (i.e., Step 2-6 in Figure 1, and (iii) Software Developer (SD), who is responsible to implement technical measures according to the chosen mitigation strategies. In this paper, we focus on the role of PE and SD, whose scenarios and requirements will be described next.

Privacy Engineer (PE). A typical privacy engineering process often begins with a PE’s review of DFD, which is created by the SA. For a more comprehensive understanding of the DFD, the PE may revisit the initial system requirements. After that, the PE can start the threat analysis, beginning with a particular element within the DFD. This analysis will involve various queries, such as "Could this data flow pose a potential threat to the system? What type of threats could emerge? If such threats exist, how can they be mitigated?".

To answer these questions, the PE explores the threat tree, which helps in identifying potential threats. Within the LINDDUN framework, seven unique threat types exist, each having its own hierarchical structure of threat trees [10]. Following the identification of potential threats, mitigation strategies can be suggested. These could involve using Privacy Enhancing Technologies (PETs) as per the existing mapping or creating a new, innovative solution beyond the current mapping.

Software Developer (SD). SDs are primarily responsible for implementing the technical measures outlined by privacy engineers. Nevertheless, SD technical expertise can contribute to a threat modeling process that includes identifying threats and selecting technical measures.

2.1. Exploratory Search Requirements

Based on our analysis on the typical scenarios of Privacy Engineering described the previous section, we identified a set of requirements to support PEs and SDs in their role in Privacy Engineering as the following:

Multiperspective Exploration. Multiple stakeholders involved in the privacy engineering process [10]. SAs provides DFDs as part of the system architecture. PEs ensure that privacy risks are identified and develop requirements for mitigating the risk. They can also recommend mitigation. The SDs will ensure that all suggested requirements and mitigations are incorporated into the system. Allowing all knowledge to be explored in one location by multiple stakeholders ensures consistency and traceability.

Data Flow Visualization. An important aspect of privacy threat modeling is understanding the data flow within a system, as this helps PEs identify potential threats. The DFD itself can be represented as a triple consisting of two elements (either external entity, process, or data store) associated with the data flow [12]. These elements also form the basis for threat identification [10]. Incorporating this data flow knowledge into the knowledge graph and visualizing it as a DFD within the exploratory search would be beneficial to the threat modeling process and provide traceability to the identified threat.

Threat Tree Visualization. Threat trees, also known as attack trees, are graphical repre-
sentations of threats or attacks against a system that are organized hierarchically. They show different ways a system can be exploited by breaking down higher-level threats into smaller, more specific threats. Therefore, a hierarchical browsing capability for threat trees is important. In addition, a text-based search would be beneficial to enable PEs to search for specific threats within the threat trees.

3. Exploratory Search Systems for Privacy Engineering

Based on the identified requirements from Section 2.1, we developed an initial Exploratory Search System for Privacy Engineering following the STAR approach [6]. We first set up the ontology and populate the knowledge graphs from existing privacy engineering datasets [10, 13, 12, 14]. Afterwards, we adapt our prior Exploratory Search systems framework [7] for this scenario.

3.1. Knowledge Graphs Construction

Ontology for Privacy Engineering. The ontology builds on concepts derived from privacy engineering methods [10, 15, 16]. The ontology aims to connect software development artifacts [13] with privacy knowledge [5]. The software development artifacts include the requirements, which may be in the form of user stories in agile requirements, and the DFD that represents them. Meanwhile, privacy knowledge might include the knowledge base about the personal data involved, privacy threats, privacy goals, legal requirements, and privacy mitigation strategies in the form of privacy design patterns. The ontology can be accessed on our GitHub page.

Ontology Population Privacy engineering tools such as ProPAN [15] or PrivacyStory [14] are examples of privacy engineering tools that would support the privacy engineering processes shown in Figure 1. These tools stored their results in various data models and formats and therefore will need to be transformed into an integrated format. In our prototype development, we use the knowledge generated by PrivacyStory and transform them into the previously described ontology. In the future, we plan to integrate more resources, e.g., the threat knowledge base and mitigation tools can be fed from the known threat knowledge base, and their mitigation can be mapped based on the known ontology [9]. Legal concepts such as GDPR can also be added to the knowledge base.

The details on the development and evaluation of the ontology and the population process from privacy engineering knowledge bases are beyond the scope of this paper.

3.2. Exploratory Search System Prototype

The implementation of the Exploratory Search System (ESS) is constructed upon the foundation laid by Haller et al. [7]. Their ESS is oriented toward the manufacturing sector. To accommodate our need for privacy knowledge within software development, we leveraged its capabilities and configured it to meet the needs of privacy engineering tasks. The implementation is accessible

---

2https://github.com/gunturbudi/ptm-ontology
3https://doi.org/10.5281/zenodo.8198322
Figure 2: Screen captures of the exploratory search systems, comprising (1) the entry area for text searches, (2) the outcome of a full-text search of the keyword "parent", (3) an information box displaying descriptions and characteristics, (4) a data flow diagram’s depiction, (5) A list of user stories related to the present selected user story, (6) a display of a hierarchical threat tree (can be accessed from the Hierarchy menu), and (7) the results of a mitigation search of the keyword "messaging".

online⁴, and a screenshot of the ESS can be seen in Figure 2. In the ESS for Privacy Engineering, we includes the identified exploration components from Section 2.1, which will be briefly explained in the following.

**Multiperspective Exploration.** Allowing all knowledge to be explored in one location by multiple stakeholders ensures consistency and traceability. The ESS system can effectively use the multiperspective exploration feature for this purpose [7]. These perspectives are intentionally designed to provide valuable insight tailored to different stakeholders without overwhelming them with information.

Figure 3 shows the perspective of PEs and SDs. The privacy engineering process begins with the selection of user stories. They can select based on specific actors or stakeholders. After choosing a particular story, the privacy engineer can view other user stories that involve the same actor. These stories come with a DFD produced by privacy engineering tools [12], providing a visual representation of how data moves through different processing. This point marks the start of exploration for threat modeling purposes.

SDs are able to see what technical measures were chosen by the PEs, while also being able to trace back why those measures were chosen and for what user stories.

**Data Flow Visualization.** We included DFD visualizations for every user story to enhance

⁴http://privacy.semantics.id
Figure 3: Exploratory scenario of PEs and SDs. PEs initiate exploration from User Story or Threat Knowledge, whereas SDs begin from technical measures. ESS, intended for exploration, draws its updates from external systems, as illustrated with threat modeling tools.

Threat Tree Visualization. LINDDUN provides a catalog of privacy threat trees grouped into seven categories of threats [10]. We’ve converted this catalog into a knowledge graph, which is displayed hierarchically in the ESS. The tree-like hierarchical representation can be accessed through the hierarchy menu, which is separate from the main text search.

The ESS also allows users to start with the text-based search to efficiently identify related topics. The underlying text search uses indexing based on the triple store. In our system, we use Lucene scoring, which uses a combination of the vector space model (VSM) of information retrieval and the Boolean model to determine how relevant a particular document is to a user’s query. Moreover, the ESS is capable of showing threats related to the currently displayed ones, thereby aiding users in navigating the threat knowledge base. Owing to its exploratory design, users have the flexibility to traverse back and forth within the interconnected knowledge base, thus expanding their understanding of related details and context. In future works, we aim to enhance this threat knowledge base by incorporating other privacy threat modeling approaches or knowledge [17].

4. Conclusion

Knowledge about privacy-by-design principles is currently scattered across different domains and locations, making it difficult for organizations to access and understand it due to the significant cognitive effort required. In this paper, we present an ontology-based search system...
to facilitate organizations’ access to privacy-related knowledge in relation to their own setting. The search system requires the organization to adapt its knowledge to the ontology and include it in the ontology that meets the standards. This system enables various stakeholders, including privacy engineers and developers, to visualize and understand data flow diagrams and threat trees through its networked knowledge base. While still in its early stages, this research lays the groundwork for future studies aimed at more effectively incorporating privacy knowledge in software development processes. It also highlights the potential of the Exploratory Search systems in reducing the cognitive demands associated with acquiring privacy knowledge.

In the future, we plan to evaluate how effectively privacy engineers or developers can use our exploratory search system to retrieve specific privacy knowledge and how they evaluate their user experience. We also plan to conduct comparative studies comparing the performance of our search system in facilitating the privacy engineering process with performance without the tools. Feedback from these evaluations will help us improve the ontology structure, user interface design, and overall user interaction flow.

Acknowledgments: This work is supported by the European Union’s Horizon 2020 research project OntoTrans under Grant Agreement No 862136.

References


A Method of Visual Presentation of Data Schemas

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

Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia

Abstract
We describe and demonstrate a method of automated creation of refined visualizations of Linked data endpoint schemas, based on their pre-computed structure information. The visualization uses UML class diagram style structure with classes, associations, and attributes, as well as subclass structure. Due to the pre-computed nature of the data schemas, visualization can avoid repeating the subclass properties and links at a superclass level and present parts of larger data schemas in a compact and conceivable way.

Keywords
Linked data endpoint schema, Visual schema, UML class diagrams

1. Introduction
Visual presentation of a knowledge graph or a Linked data endpoint schema can be expected to help a user to comprehend the graph/endpoint structure and, therefore, use more efficiently the data contained therein. There are several tools allowing visualization of existing Linked data endpoint schemas, such as LD-VOWL [1] and LODSight [2], allowing to obtain on-the-fly visualizations of the class-to-property relations actually present in the suitably sized data sets.

The actual data schema information, important both for visual characterization of the data set contents and for technical assistance in handling the models based on such a data schema (e.g., by providing context-aware model auto-completion) is, however, much richer than the class-to-property correspondence alone. Such a schema can distinguish between more relevant and less relevant classes as property sources and targets (as described further on in the paper), it also can involve e.g., property domain/range information (in the ontological sense), and cardinalities.

The means for knowledge graph schema description are provided also by RDF data shape languages SHACL [3] and ShEx [4]. The concepts used in the data schema can also be described by means of OWL ontologies [5].

Due to the richness of the information the data schemas may contain, it might not (and typically would not) be possible to obtain their full information on-the-fly, with the user waiting behind the computer screen. In this demonstration we work in the setting of creating a data schema first and further on providing its visualization in the form of an extended UML class diagram, paying attention to the features we find necessary for the successful schema presentation.

The primary use case for the data schemas we consider here is serving the context-aware auto-completion in building visual queries over RDF data in the ViziQuer tool [6]. Still, data schemas are important also besides the particular tool. The ability to present the data schema visually to the users that seek to build visual data queries over the schema would be an important contribution towards a more encompassing visual experience in their data analysis work.

There exists a wealth of tools for visual presentation of OWL ontologies, including VOWL [7], OWLGrEd [8] and OntoDia [9]; [10] provides an expanded survey of such tools. There are tools for RDF data shape visualization, as well, including, e.g., RDFShape [11]. The data schema visualizations we provide here have a common conceptual understanding with the existing OWL and RDF data shape visualization methods by using graph of class and property connections as
the visual data schema image. Our presentation differs both from the OWL ontology and ShEx/SHACL shape presentations in that it concerns the actual data structure, as it is present in the data endpoint, and it involves a focus on important nuances (as the relevance of a class as a property source or target) that are not present or are present partially (e.g., as designating classes as property domains/ranges) in the existing visualization tools.

In the context of RDF data summarization (cf. [12] for a survey), our work is primarily concerned with fine-grained presentation of the entities (classes and properties) of the actual data schema that can be acted upon during the query creation.

In the rest of the paper, Section 2 provides a motivational example and the data schema concept description, Section 3 describes the visualization of the data schemas, and Section 4 provides brief conclusions and outlines directions for future work.

2. Data Schemas

We consider data schemas that are based on property availability at classes and connections of classes by properties, including the situations when a property can link multiple pairs of classes. Should it be typical for an instance in a data set to belong to multiple classes (including, but not limited to, the case of a subclass and a superclass), a direct class-to-property mapping may exhibit a set of links that is overly large for characterizing the actual connections of instances in the data set. This would show up immediately in the schema diagram visualization, where the extra links both overload the schema and may even obscure the actual connections.

As a simple example, consider the data set of Nobel Prizes (we work with a copy made for easier connection reasons). Figure 1 shows a (manually drawn) fragment of the data set schema, regarding Awards, Laureate Awards, and Nobel Prizes; similar structure of links is obtained also in the LD-VOWL visualization. Here only the two direct links between nobel:LaureateAward and nobel:NobelPrize would be explicitly relevant; all links involving the dbo:Award class would actually obscure the data set presentation.

![Figure 1: Naïve links in Nobel Prizes data set schema](image)

If the data schema is used to support a modeling environment (e.g., in context-dependent data query completion, either in SPARQL form, or possibly in visual form, as in ViziQuer tool), the full list of source and target classes for a property would need to be collected. However, the markers for the class importance in the context of a property would be essential also in the modeling environment. For instance, if a class were the property domain or range (the smallest domain/range, if there are several ones), this would allow to consider all other source/target classes for the property to be less relevant and even auto-complete the property domain/range information. Still, the data schemas need to also cover the cases where there are more than one relevant class designated as a source or target class for a property (intuitively, the union of all “relevant” classes would need to cover the entire set of subjects or objects in the property triples).

---

2 https://data.nobelprize.org/sparql
5 Here and further on, the used namespace prefixes stand for their usual URIs. The prefix nobel: stands for http://data.nobelprize.org/terms/ and ncat: stands for http://data.nobelprize.org/resource/category/
6 http://viziquer.lumii.lv/, https://viziquer.app
Figure 2 outlines the structure of (an essential fragment of) the data schema used for storing the class-to-property correspondence, together with the described relevance markers (cover_set_index in the cp_rels table that relates the classes and properties) also stating the relation type_id (1 for incoming and 2 for outgoing property), cardinalities and property domain/range class information. The schema contains also further information that is used in the setting of auto-completing visual queries over RDF data in the ViziQuer tool, although the schema structure and its data can be used also independently of the visual tool.

Figure 2: Data set schema structure

We offer the data schema filling by running first the open-source OBIS Schema Extractor tool\(^7\) that issues a (possibly, large) series of SPARQL queries over a data set and creates a JSON-encoded structure of the data set schema (there are options to specify the level of granularity of the information to be retrieved from the data set at the schema extractor interface). The created JSON file is then stored into the database of the described schema by the data shape import service tool\(^8\), followed by some manual editing of the namespace prefix abbreviations (e.g., for namespaces not found on prefix.cc site).

3. Schema Visualization

The data schemas described in Section 2 assist query auto-completion in the ViziQuer tool (with possible independent usage). The work demonstrated here shows the possibility to visualize the corresponding schema in form of extended UML class diagram thereby providing the visual schema presentation in addition to the visual ViziQuer query creation tool environment.

The idea of the schema visualization is to ascribe the properties only to those source and target classes that are relevant in the sense described in Section 2, so not ascribing a property link source/target to a superclass, if all property subjects/objects belong to a subclass, and not ascribing a property link source/target to a subclass if it is ascribed to a superclass.

---

\(^7\) https://github.com/LUMII-Syslab/OBIS-SchemaExtractor
Figure 3 shows a visualization of the Nobel Prizes data set schema in accordance with these principles. We note the single appearance of the properties dct:hasPart and dct:isPartOf connecting the nobel:LaureateAward and nobel: NobelPrize classes, in contrast to the naïve visual schema presentation in Figure 1.

Should only the object properties connecting the classes and the data properties at classes been visualized, an important property nobel:category would have been missing from the schema, as it connects (in the actual dataset) the instances of nobel:LaureateAward and nobel: NobelPrize classes to resources that do not have their class assertions specified. Therefore, a design is suggested to show an object property (nobel:category in the example) at a source class as an attribute, if some of its object instances are classless.

The created data schema is further on enriched by the property triple statistics (optional) showing the property triple count in the context (source class or source and target class combination) and total property triple count, property domain (D) and (local) range (R) information (in the ontological sense) and property max cardinality (1 or *).

To offer a further compactified data schema presentation, abstract superclasses (as dbo: City or dbo: Country in the example) can be introduced into the data schema, based on the shared properties for which the classes are sources or targets. An abstract class is introduced based on shared incoming and outgoing object properties, however, when introduced, it lists the data properties common to all its subclasses, as well. The ontological range information for properties incoming into the abstract superclass is not shown in the example schema, however, it can be computed (approximated) from the information that is collected in the data schema (Figure 2).

Figure 3: Nobel Prizes data set schema

The data schema shown in Figure 2 (in Section 2) admits a possibility to specify a classification property (classification property field in table classes) used for identifying a class in the data. The typical classification property is rdf:type, however, there are endpoints that use different classification properties (e.g., Wikidata\textsuperscript{10}). In the considered Nobel Prizes data set example the classes in Figure 3 are created each with the rdf:type property, however, it would be possible to

\textsuperscript{9} Since the abstract superclasses currently are not reflected in the data schema (Figure 2), the statistics information for the properties at an abstract superclass may be higher than the actual numbers, if there are individuals shared by

\textsuperscript{10} https://query.wikidata.org/
introduce into the schema also the classifiers corresponding to other properties. Figure 4 shows such an extended schema, where, in addition to the "standard" classes, the classifiers, corresponding to the nobel:category property, are shown.

In the example the classifier values corresponding to the nobel:category property are marked with the prefix (ct), and are shown using a simplified notation (list of multiple class names/classifier values in a single visual container) since none of them have any incoming or outgoing property characteristics that would be different from their container superclass.

Note that the categories for Nobel prizes and Laureate in the actual data set, although have the same local names, are different resources that belong to different namespaces (and therefore are to be listed separately in the extended data schema).

![Figure 4: Extended example schema, with category classifiers](image)

The visualization module is freely available\(^\text{11}\), together with its usage instructions in the ViziQuer tool context. The visualization starts from an existing ViziQuer project, from which the data to be loaded into the visualization tool are generated (an independent use of the visual schema presentation module would require generating this information in some other way).

The technical implementation of the visualization is currently performed within the desktop based GrTP/TDA platform \(^\text{13}\) (the one providing the framework for the OWLGrEd visual ontology editing tool\(^\text{12}\)). It is one of avenues of future work to integrate the visual schema presentation into the web-based environment of the ViziQuer tool.

The visualization module repository also provides links to the information on the currently visualized schemata. The current schemata list involves schemas over Nobel Prizes, Mondial\(^\text{13}\) and ISWC 2017 metadata\(^\text{14}\), and is kept growing.

\(^{11}\) https://github.com/LUMII-Syslab/dss-schema-explorer
\(^{12}\) http://owlgred.lumii.lv
\(^{13}\) http://servolis.irisa.fr:3232/mondial/sparql
\(^{14}\) http://servolis.irisa.fr/iswc2017/sparql
4. Conclusions and Future Work

In this work we have demonstrated that a saved data schema, in particular the one used by the ViziQuer tool environment, may contain useful information for the data schema visualization that would typically not appear in the simple class-to-property mappings that could be retrieved on-the-fly from the suitably sized data endpoints.

The concept of a data schema expansion to introduce non-standard classifiers on the level of the data schema and its visual presentation has been introduced, as well.

The principal directions of the future work include a fine-tuning of the visualization concept (e.g., by dealing with properties whose triple subjects or objects are classless). The concepts and algorithms from RDF graph summarization area (cf. [12]) can be expected to be helpful here.

Regarding the use of the visualization in the ViziQuer tool the principal future work avenues include integrating the visualization solution into the web_based tool environment, developing the options for schema fragment presentation and navigation, as well as enabling the data query creation over the data schema backbone presentation (since both the data schema presentation and a developed visual query would have similar UML-style diagram structure).

Acknowledgements

This work has been partially supported by a Latvian Science Council Grant lzp-2021/1-0389 “Visual Queries in Distributed Knowledge Graphs”.

References

Tree Visualization of Patient Information for Explainability of AI Outputs

Sandeep Ramachandra\textsuperscript{1,∗,†}, David Vander Mijnsbrugge\textsuperscript{1,†}, Pieter-Jan Lammertyn\textsuperscript{2}, Stijn Dupulthys\textsuperscript{2}, Femke Ongenae\textsuperscript{1} and Sofie Van Hoecke\textsuperscript{1}

\textsuperscript{1}IDLab, Ghent University-imec, Ghent, Belgium
\textsuperscript{2}Radar, AZ Delta, Roeselare, Belgium

Abstract

Knowledge graphs (KG) and ontologies can be leveraged to efficiently convey information and provide a great aid in explaining the outcomes of neural networks in the healthcare domain. In this short research paper, we introduce a novel approach that encodes patient information and expert knowledge of diseases into a single temporal graph which enables seamless integration into neural networks. Furthermore, we present a visualization tool that explains the output generated by these networks, leading to better understanding of the provided decisions by healthcare professionals and other stakeholders.

Keywords

Explainable AI, Visualizations, Dynamic graphs, Temporal graphs

1. Introduction

Patient health records are multi-modal in nature, containing a combination of structured data (e.g., performed procedures and made diagnoses), free text (e.g., doctor notes), and higher dimensional data, such as images (e.g., X-ray and CAT scans) or time series (e.g., heart beat measurements). Moreover, patient information is dynamic in nature and constantly evolves over time, e.g., changing laboratory test results over subsequent hospital visits, and/or the condition of the patient changing over time. This makes patient information an inherently complex data source to work with.

To enable proper analysis, e.g., with Machine Learning (ML), it is important to be able to represent this multi-modal data in a data structure which makes the correlations between the data explicit. Due to their expressive power, Knowledge Graphs (KGs) and ontologies have become increasingly popular to encode such patient information [1]. Moreover, their graph representation allows visual and easily understandable interpretation of the information. An ontology enables us to additionally encode the established expert-knowledge about the...
healthcare domain and connect it to the data in the KG, allowing for the incorporation of correlations between symptoms and diseases in the KG. Leveraging the incorporated prior knowledge empowers more effective data analytics and enhances performance. On the other hand, the utilization of timing information in KGs, typically represented using date time nodes attached within the same graph, proves less optimal when analyzing patient information, as the timing information represents a change in the graphs itself. In order to facilitate this interpretation, the patient information can be represented as graphs that dynamically change over time rather than using time nodes in the complete graph.

At the same time, there has been a surge in the popularity of machine learning (ML) methods and graph embedding techniques that empower the execution of advanced data analytics using these KGs. Prominent examples include Graph Convolutional Networks (GCNs) and RDF2Vec [2]. In critical domain like healthcare, eXplainable AI (XAI) is gaining increasing importance [3]. As any erroneous output can be harmful to the patient, the healthcare expert, and potentially other non-AI experts (e.g. care provider, patient), should be able to clearly trace the significant input features contributing to the output of the ML. Furthermore, it is essential that this explanation aligns with, or is substantiated by, the expert knowledge of the domain, which is captured in the ontology. While some recent graph embedding methods try to retain the interpretable aspects of KGs, e.g. INK [4], most graph embedding methods are considered black-box, especially the ones based on deep learning techniques, such as GCNs. The latter are therefore often combined with post-hoc explanation methods aiming to elucidate the model’s output for specific inputs, such as SHAP[5] and Saliency Maps[6]. To do so, these methods offer insights into the contribution of each input feature towards generating the final output, and thus into an importance factor of each feature. This feature contribution can also be used in KGs to visualize the importance factor of each node in the KG using a color scale over the post-hoc explanation of the output of the AI for any given task. Prior works on visualization and post-hoc explanations for KGs do not take into account the dynamic temporal patient data [1]. Therefore, in this short research paper, we propose a novel methodology to encode dynamic temporal patient information in a KG and to visualize the output of the graph embedding model in a clear and understandable manner by visualizing the real world patient information along the time axis as well as visualizing the importance factors for each entity in the patient graphs for ease of comparison for a healthcare expert.

2. Related work

There are two domains coming together in this paper, namely patient information encoding in a KG optimized for data analytics, and graph visualization optimized for exploration by domain experts interpreting the output of these analytics. We therefore explore the state-of-the-art in these two domains below.

**Graph encoding of patients** With the design of ontologies, such as SNOMED [7] and OMOP [8], a lot of work has already been performed on representing patient information in KGs. However, most of these representations ignore the temporal dimension of patient data and fail to fully harness the intrinsic explainability offered by graph representations. In Choi et al. [9], the authors recognised the EHR data as being multilevel with diagnosis being related to certain
treatments and utilise this property to improve the performance of their model. However, most concepts are interrelated to each other and lack fixed hierarchical levels. This shortcoming is solved by [10] with the representation of patient visits being in graph structures. In [11], the authors combine a patient graph with ontologies to improve the quality of the embeddings of the concepts. These papers show the potential of graph neural networks with patient data but fail to explain their output which is critical in the healthcare domain.

**Graph visualization** There is no one size fits all solution for graph visualization [12]. Every use case for ontology and KG visualization utilises different tools and focuses on different aspects of visualization. For example, VOWLExplain [13] utilised WebVOWL [14], a web based ontology visualization tool, to visualize patient information from The Cancer Genome Atlas and visually explain the recommendations of an AI model. Their user study showed that the graph explanations of AI recommendations regarding the patient were equally accurate and comprehensible compared to textual explanations. While the paper showcases the explainability inherent to KGs, they did not explore the temporal aspect of patient information. In [15], the Neo4j graph database is explored for a healthcare case where they showcase the patient’s progression along the time axis. The presentation was largely exploratory and did not use the explainability offered by graphs. So to conclude, state-of-the-art focuses on either the temporal KGs or on explainability, where we are the first to tackle the combination of both.

3. **Methodology for representing and visualizing patient information in a KG optimized for explainable AI**

In this section, we first highlight the requirements for the proposed KG representation method to enable XAI with Graph Neural Networks (GNN), and then dive deeper into the proposed methodology itself and the accompanying visualization.

3.1. **Requirements**

The requirements for the encoding of a patient graph are:

- **Dynamic Temporal view**: The patient graph should be separable into different time steps since patient data is added *incrementally* over time, e.g. per visit to the hospital. Real-world patient data is constantly updated according to the changing condition of the patient and performed procedures, e.g. new laboratory results, change in medication, and new diagnoses. The visualization should reflect this. Furthermore, this incremental nature should result in a much less cluttered representation of the data, especially after prolonged periods of time.

- **Tree view**: Ideally, the patient data is structured in a tree-based manner as this accurately reflects the observatory nature of the data. Namely, each specific diagnosis, treatment, observation, laboratory observation, etc. should be *unique* in the representation, as they are likely to be *repeated* across visits, e.g. chemotherapy is a treatment that is repeated many times over a given period where each instance of the treatment carries relevant information. The patient itself represents the trunk of the tree. Each visit then represents a branch in the tree, with all the data connected to that visit, e.g. observations, treatments,
3.2. Methodology

Our novel method to generate a patient graph containing the needed information meeting the requirements set out in Section 3.1 consists of the following 4 steps:

1. **Triple generation**: As a first step, the multi-modal data has to be collected from their original representation, e.g. a relational database or data lake, into an intermediary representation so that one patient’s data (Conditions, Treatments, Drugs, Labs, Observations) is encapsulated in one object along with their associated timestamp. Each concept node has to be made unique so that the nodes do not get conflated in the graph (See Tree requirement in Section 3.1). As ontology, the OMOP (Observational Medical Outcomes Partnership) [16] ontology is used. Using relationships set out in this OMOP schema, information on each object is converted into a N-Triples (NT) file that is used to generate the patient graph. Figure 1a shows a toy example for this triple generation.

2. **Time separation**: Next, each timestamped concept present in the generated NT file has to be separated along the time axis. To do so, each time step can be defined as either the time when a discrete change in patient data occurs, or as a fixed change in time. For this paper, we chose discrete changes as this way the changes are immediately apparent.

and diagnoses made, as subbranches within that branch resulting in a hierarchical tree structure with variable length branches.

- **Ontology**: It is important to be able to link the data in the KG to the prior knowledge encoded in the ontology, as this delivers important information to the XAI. For example, each diagnosis and treatment in the procedure are linked to each other in the ontology.
Discrete changes are also better when dealing with patient data that extends over years, for example for chronic (long term) diseases, in the NT file for each patient, we then create a new NT file for the new graph per time step. A quad file can also be created with each time step as a disjointed graph as this would behave in a similar manner. Figure 1b continues with the toy example showing a visualization of the graph separated on the time dimension.

3. Adding ontology: The ontology of OMOP is contained in two tables in OMOP schema, i.e. Concept Ancestor and Concept relationship, with both incoming and outgoing relations defined. To comply with the Tree requirement (see Section 3.1), only the outgoing relations are taken. In this step, from each patient’s NT file, the leaf nodes are used to look for matching subject nodes in two tables. The filtered triples are connected to each leaf node in patient graph, duplicated across the time axis.

4. Visualization: As the visualizations have to be dynamic and compatible with the frameworks of the neural networks, we chose Plotly [17] due to its interactive features and ease of use to enable the time axis visualization in the fourth and final step. To do so, each time step of the patient graph is added as a trace to the Plotly figure. Since the patient graph is modelled as a tree, we use the Reingold-Tilford layout to spread out the nodes and layer of the tree for better visibility. Additional information about each node is added to a hover interaction with the node. For visualizing the explanation, the contribution of each node to the output of the neural network can be calculated/inferred. For example, attention blocks [18] can be used to gather this information. As such, an importance is calculated for every node which can be assigned as a color to each node, allowing for instant visibility of the importance of each node at each time step. One can also highlight the time step with the most contribution by surfacing it first while viewing the visualization. This can be interesting, for example, for healthcare professionals.

4. Use case: Lung cancer patient representation and visualization

We applied the presented method on a synthetic lung cancer patient, designed together with clinical experts from AZ Delta, to empirically prove applicability of our method.

Figure 2 shows a screenshot of an interactive example patient visualization developed using the methodology presented in Section 3.2. Due to the example being synthetic patient data, the importance factors are assigned randomly. A live example can be found at https://predict-idlab.github.io/Tree-Visualization/. The example shows an overview of the patient information as well as the sub-graphs for related concepts, extracted from the OMOP ontology and attached to each instance of the concepts. The example visualizes the patient KGs with the temporal information retained and showcases the explainability possibilities of the patient KGs. The visualization clearly displays the most important nodes in a darker shade of red so that a healthcare professional can, without any deep understanding of the model used, immediately notice the most significant node to the model at that time step. The same conclusion can be made across the time dimension as the time step that is the most significant, can be highlighted by displaying it by default.
5. Conclusion

We have proposed a novel methodology for the representation and visualization of patient information from multi-modal patient information while also incorporating the time dimension. Bringing this time dimension to patient representation and visualization is important to enable us to utilise the dynamic nature of the patient data. The resulting representation can be used in a Graph Neural Network (GNN) for various downstream tasks. Our method explains the output of this GNN using importance factors which are incorporated in the visualization as node coloring. This provides a visually easily interpretable explanation of the GNN output. We showcased the correct functioning of our methodology on a lung cancer use case, using synthetic patient data. In the near future, we hope to add interactive views of the graph with highlighting paths for important nodes and real time visualization of graphs.

Acknowledgments

This study was partially funded by the Flanders AI Research Program and the VLAIO O&O ADAM project with AZ Delta (HBC.2020.3234).
References


Visualizing Ontology Metrics In The NEOntometrics Application

Achim Reiz\textsuperscript{a} and Kurt Sandkuhl\textsuperscript{a}

\textsuperscript{a} Rostock University, 18051 Rostock, Germany

Abstract

Metrics offer an objective assessment of the inner fabrics of ontologies. They allow us to quickly understand graph properties, logical complexity, completeness of human-centered annotations, or degree of interconnection. When analyzed historically, ontology metrics tell much about development decisions and the impact of changes and can be used for quality control measures. The NEOntometrics software allows calculating evolutional ontology metrics for git-based repositories and implements the majority of literature-proposed metric frameworks. This paper presents the recently added visualization capabilities for visualizing the differences between ontologies in a repository, assessing the change impact of the most recent commit, and examining ontology evolution.

Keywords

NEOntometrics, Ontology Metrics, Knowledge Graph, Quality Management

1. Introduction

Ontologies underpin the technology of applications such as question-answering systems, recommendation systems [1], or autonomous driving [2]. They build on subsets of first-order logic, which has little in common with traditional data modeling or imperative programming. The development team is often dispersed skill-wise, experience-wise, in their responsibilities, and geographics [3]. The complexity and collaborative development processes put quality control activities at the forefront. Users and ontology managers need to understand the impact of proposed changes and the evolutional history of the artifacts as a whole to make informed development decisions.

One way to gather these kinds of information is the calculation of metrics. Ontology metrics translate the structural attributes of ontologies into objective, reproducible measurements. The measures cover the use of RDF(S) and OWL formalisms, graph properties, or human-centered annotations.

With NEOntometrics [4], a tool is available for analyzing large amounts of historical metric data based on git repositories. NEOntometrics implements most proposed ontology metrics and allows csv data export or integration into custom applications and analyses using a GraphQL interface. Until recently, however, the frontend visualization capabilities were limited to a tabular metric representation.

This paper presents a new integrated visualization feature for NEOntometrics. The evolutional developments and the differences of ontologies in a repository can now be displayed using diagrams. A comparison view for the last two versions of an artifact visualizes changed measures, thus allowing a quick overview of impacted structural attributes. Integrating diagram capabilities into software for calculating ontology metrics eases the consumption and productive use of the measures and hopefully contributes to the broader dissemination of ontology metrics for quality control.

The paper is structured as follows: The next section recapitulates the state of the art regarding ontology evolution and visualization approaches. Section three presents NEOntometrics, including the newly implemented visualization features. The research concludes with an outlook and a discussion.
2. Related Work

In this related work, we aim to recapitulate the current state regarding software support for ontology evolution and evolution visualization. [4] further contains a review of software support for ontology metric calculation. Hence, it is not thoroughly discussed in this paper.

Analyzing and visualizing ontology evolution is a multifaceted field with various manifold approaches. As such, there are already extensive literature reviews that collected the relevant state of the art:

Novais et al. [5] collected an extensive review of visualization approaches in the related field of software evolution and classified the existing research (among others) along their application scenarios, visual paradigms and attributes, data sources, and explanation strategy.

Lambrix et al. [6] collected software for ontology and software evolution and categorized them along their functionality, which is further assigned to evolution steps and change categories. De Leenheer and Mens [7] presented processes and tool support for single-developer and collaborative ontology development processes. They focus on the challenges of inter-organizational changes with multiple possible ontology managers and the corresponding organizational requirements. These authors also present their own collaborative ontology engineering process DOGMA, including tool support [8].

Tudorache [9] recapitulated the recent advances in ontology engineering, including software support. While she argues that the tool support is nowadays much better suited not only for research but also for commercial projects, she also argues for making access easier for newcomers and for increasing the usability of the tools.

### Table 1

<table>
<thead>
<tr>
<th>#</th>
<th>Software</th>
<th>Type</th>
<th>Use case</th>
<th>Available</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10]</td>
<td>ReX</td>
<td>Standalone</td>
<td>Identify and visualize unstable ontology regions</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>[11,12]</td>
<td>CODEX, ContoDiff Explorer</td>
<td>Standalone, API</td>
<td>Understand and visualize ontology changes and their impacts</td>
<td>❌</td>
<td>✔️</td>
</tr>
<tr>
<td>[13]</td>
<td>Ecco</td>
<td>Shell</td>
<td>Detect changes between two ontology versions</td>
<td>❌</td>
<td>✔️</td>
</tr>
<tr>
<td>[14]</td>
<td>PromptDiff</td>
<td>Protégé Plugin</td>
<td>Comparing Structural changes between two ontology versions</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>[15]</td>
<td>OWLDiff</td>
<td>Standalone, Protégé Plugin</td>
<td>Compare and merge two OWL2 ontologies</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>[16]</td>
<td>ChIMP</td>
<td>Protégé Plugin</td>
<td>Dashboard view of evolved ontology metrics</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>[8]</td>
<td>Dogma Studio</td>
<td>Standalone</td>
<td>Manage ontology evolution in collaborative environments</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>[17]</td>
<td>Live Diff Taxonomy</td>
<td>Protégé Plugin</td>
<td>Summarizes the changes in taxonomical view</td>
<td>✔️</td>
<td>(✔️)</td>
</tr>
</tbody>
</table>

Table 1 presents tools for detecting and visualizing ontology evolution. Many activities have been at the automatic detection of differences between two ontologies with implementations in [12–15]. These tools detect the differences between two ontology versions and return the changes mainly as a list. Graphics-wise, [11] offers a visualization of the ContoDiff algorithm, [10] allows visualization of

---

2. https://github.com/dbs-leipzig/conto_diff – Only the underlying algorithm of CODEX is open source
3. https://github.com/rsgoncalves/ecco/
unstable ontology subparts. To instantly understand the impact of modeling decisions in the Protégé editor, [17] calculates a taxonomy of changes, and [16] calculates numerical differences between the saved version and currently performed changes.

While the visualization view of the new NEOntometrics features (cf. Figure 4) has some similarities to [16], our approach does not calculate the metrics instantly for a change but regards the last two committed versions in a git repository. NEOntometrics also implemented a more significant number of ontology metrics, allowing us to choose from ~160 metrics from different frameworks. Compared to the current state of the art regarding tool-supported ontology evolution, the NEOntometrics approach is strictly focused on visualizing ontology metrics. Further, it not only regards two versions but calculates and visualizes the overall version history, thus allowing conclusions on overall evolutional processes and design decisions.

3. NEOntometrics

NEOntometrics is a web-based application that calculates evolutional ontology metrics of git-based repositories. It iterates through a repository and calculates the respective measures for every ontology in every commit if the file has changed. It comes with an interactive help page Metric Explorer, is open source\(^7\), and is available online\(^8\). For more information on NEOntometrics, cf. [4,18].

As calculating the version history of ontology repositories can take a considerable amount of time, NEOntometrics works asynchronously. A new calculation can be triggered by pasting the URL to a given repository in the bottom text field of the Calculation Engine (cf. Figure 1). If the repository is not yet known to the system, it can be put in the calculation queue. Afterward, a separate worker application retrieves the ontology from the git repository and starts the analysis.

The ontology metrics can be retrieved as soon as the calculation is finished. Currently, NEOntometrics supports OQuaRE [19], OntoQA [20], oQual [21], the cohesion metrics by Yao et al. [22], the good ontology metrics by Fernández et al. [23], Orme et al.’s evolutional metrics [24], and the complexity metrics by Yang et al. [25]. Metrics can be run on the ontology as it is or on the inferred graph. However, as the inference engine can take a considerable time, it is only advised for small ontologies.

![Figure 1: The entry point of NEOntometrics for the metric calculation. The new visualization feature is called using the “show the analytic” button.](https://github.com/achiminator/NEOntometrics)

---

7 [https://github.com/achiminator/NEOntometrics](https://github.com/achiminator/NEOntometrics)
8 [http://neontometrics.com](http://neontometrics.com)
After selecting the desired metrics and frameworks and triggering the retrieval process, the software shows a paginated tabular view of the ontology metrics. The button “show the analytic” opens the visualization page. The measures shown in the diagrams are always congruent to the ones selected during the ontology retrieval process.

Three visualizations are available to examine the most recent changes, compare ontologies in a repository, and visualize the ontology evolution. The figures below exemplify these visualizations for the SciData ontology\(^9\) for interoperable scientific data exchange [26].

**Figure 2:** Bar chart for comparing the ontologies in the repository in their recent version for the selected ontology metrics.

The first chart (cf. Figure 2) displays differences between the repositories’ ontologies in a bar diagram. The bar diagram presents the last version of each ontology. The diagram can be scrolled by dragging the picture to either side of the frame. Hovering over a measure shows the detailed measures. Ontologies can be selected and deselected by clicking on their name below the chart. The scaling changes dynamically depending on the sizes of the bars visible in the frame.

The bar chart in the example visualizes the *axioms* and *classes* of the repositories’ ontologies. The *thermo*, *scidata*, and *cao* files have the most classes and axioms. However, *cao* has many more axioms than *thermo*. Thus, one can conclude that the *thermo* ontology is more driven by a taxonomical class structure, while *scidata* and *cao* have more additional axioms than *thermo*.

Further examination revealed that *cao* and *scidata* are indeed more logically interconnected. In the given example, the bar chart visualizes discrete count-based measures. However, the bar chart visualization works also for ratio-based formulas. For example, the OntoQA metrics shown in Table 2 and visualized in the line chart in Figure 3 could also be used in the bar diagram.

The second visualization (cf. Figure 3) contains an evolutional view of one ontology for the selected ontology metrics. It allows an understanding of the impact of changes on the structural attributes and can be used to understand and evaluate design decisions. The drop-down menu at the top of the window changes the ontology to be visualized. The user can activate and deactivate measures by clicking their name on the legend below the chart, and the diagram scales automatically, similar to the bar chart. Hovering over a data point gives further details on the displayed value.

The diagram in Figure 3 shows four OntoQA measures for the *scidata* ontology. At first, it is evident that relationship diversity is constantly at 0, originating from the fact that no object properties are declared on classes. The class utilization and average population increase early in the lifetime of the ontology. Otherwise, they stay reasonably consistent. No individuals nor classes were added in this ontology, and the changes in classes and the subclass structure are relatively subtle. On the opposite,
the attribute richness increases heavily. As the classes are somewhat consistent, these changes are driven by data and object properties. Also, the ontology describes a relatively high number of data and object properties compared to declared classes.

Table 2
The OntoQA measures visualized in Figure 3

<table>
<thead>
<tr>
<th>Measure</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Richness</td>
<td>(\frac{\text{dataProperties} + \text{objectProperties}}{\text{classes}})</td>
</tr>
<tr>
<td>Average Population</td>
<td>(\frac{\text{individuals}}{\text{classes}})</td>
</tr>
<tr>
<td>Relationship Diversity</td>
<td>(\frac{\text{objectPropertiesOnClasses}}{\text{subClassOfAxioms}})</td>
</tr>
<tr>
<td>Class Utilization</td>
<td>(\frac{\text{classesWithIndividuals}}{\text{classes}})</td>
</tr>
</tbody>
</table>

Figure 3: Line chart for one ontology showing the complete evolitional process.

The last analysis shows which measures have changed and by how much in the recent commit. It thus provides a detailed view of the ontologies’ last two versions and shall allow a quick assessment of the impact of the most recent change to evaluate the usefulness and identify eventually unintended side-effects. Similar to the previous analysis, the ontology can be selected in the drop-down menu at the top of the application. In contrast to the other visualizations, this view shows not only the selected metrics but every metric that has changed. The little icon on the left indicates an increase (⬆️) or decrease (⬇️). If a metric does not change, it is not on the list.

Figure 4: Summary outlining the changes between the last two versions for one ontology
4. Conclusion

The evolutional analysis of ontology metrics offers an objective insight into the evolvement of their structural attributes and can tell much about underlying design decisions. With the software NEOntometrics, tool support is available for analyzing large quantities of ontologies. The new visualization capabilities presented in this paper extend the application for an easily consumable human-oriented metric interface. We hope they ease the consumption of ontology metrics and contribute to a broader dissemination of metrics for quality control.

Application for the ontology metrics are manifold. For example, the metrics allow the user to understand evolutionary processes or differences between various ontologies. Ontology metrics can aid in making better-informed reusing and development decisions as they allow quickly grasping an ontology’s inner fabrics. One can set objective and reproducible goals for ontology developments and track their achievement through metrics.

Soon, we plan on integrating even more visualization capabilities to allow a better deep dive into the ontology development processes. Further on the NEOntometrics roadmap is integrating more ontology metrics, e.g., to assess SHACL-constructs and use of custom vocabularies. In that sense, we are interested in the functionalities that the community would like to see implemented. A potential evaluation of the usefulness of the visualizations and metrics is also desirable in future work.

5. References


