Visual Presentation of SPARQL Endpoint Data Schemas

Experiment protocol

Experiment performed by: Jana Fedotova

With supervision by: Kārlis Čerāns

Introduction

The following protocol outlines the procedures and methodologies for conducting an experiment focused on the creation of aesthetically pleasing data schemas from SPARQL endpoints using OBIS Shema-extractor and ViziQuer. SPARQL (SPARQL Protocol and RDF Query Language) is a powerful query language used to retrieve and manipulate data stored in RDF (Resource Description Framework) format. While SPARQL is proficient in data retrieval, the presentation of data often lacks visual appeal and may pose challenges for interpretation by non-technical users.

This experiment addresses the need for visually appealing data schemas that facilitate comprehension and communication of complex semantic structures derived from SPARQL queries. By transforming raw data into "beautiful" data schemas, we aim to enhance the accessibility and usability of RDF data for a broader audience.

The creation of aesthetically pleasing data schemas holds significance in various domains, including data visualization, knowledge representation, and semantic web applications. Visual representations of data schemas can facilitate data exploration, analysis, and decision-making processes, ultimately leading to more informed insights and actions.

This protocol describes a systematic approach for generating visually appealing data schemas for SPARQL endpoints.

Hypothesis

There exists a method for retrieving and visualizing schemas for SPARQL endpoints of small to medium size.

Materials

Tools used for experiment:

1. OBIS-SchemaExtractor (https://github.com/LUMII-Syslab/OBIS-SchemaExtractor) - OBIS Schema Extractor is Java based web application - REST controller to process data schema extraction from SPARQL endpoints or OWL/RDF files.

2. Data Shape Server (DSS) (https://github.com/LUMII-Syslab/data-shape-server#data-shape-server-dss) - The Data Shape Server serves a stored knowledge graph schema information.
3. ViziQuer (https://github.com/LUMII-Syslab/viziquer/tree/development) - provides visual/diagrammatic environment for ontology-based data query definition and execution.
4. DSS-schema-explorer (https://github.com/LUMII-Syslab/dss-schema-explorer) - allows you to get a visual representation of a DSS schema.
5. PostgreSQL.

Data Collection

The data collection phase of this experiment involved gathering RDF (Resource Description Framework) data from three distinct sources:

- Linked Open Data cloud [1],
- ISWC 2023 Proceedings [2] [3],
- Inria catalog[4] .

The data collection and filtering procedure aimed to gather relevant statistics from RDF data endpoints and subsequently apply criteria to select suitable endpoints for further analysis. The collected statistics include class count, property count, and triple count from each endpoint. Additionally, endpoints were filtered based on three criteria: class count, property count, and endpoint activity status.

Collected statistics:

- **Class count:** The number of distinct classes or types of entities present in the RDF data.
- **Property count:** The total number of distinct properties or relationships between entities.
- **Triple count:** The total number of RDF triples, which consist of subject-predicate-object statements.

Endpoint filtering criteria:

- **Class count is less or equal than 50:** Endpoints with a class count more than 2 and less than 50 (or 51 in specific cases if the author considered this data interesting) were selected. This criterion ensures that endpoints contain a

[1] https://lod-cloud.net/lod-data.json
[2] https://link.springer.com/book/10.1007/978-3-031-47240-4
[3] https://link.springer.com/book/10.1007/978-3-031-47243-5
[4] http://prod-dekalog.inria.fr/sparql

manageable number of distinct classes, facilitating subsequent analysis and visualization.

- **Property count is less or equal than 2000:** Endpoints with a property count less than or equal to 2000 were considered. Limiting the number of properties helps focus the analysis on endpoints with relatively simpler data structures.
- **Active endpoint status:** Only active endpoints were included in the analysis. Active endpoints are those accessible and responsive to queries at the time of data collection, ensuring the reliability and availability of data for analysis.

Procedure:

1. Data collection involved querying each RDF endpoint to retrieve statistics on activity, class count, property count, and triple count.
2. Collected statistics were stored in a structured format for further analysis.
3. Endpoints were then filtered based on the specified criteria using automated scripts or manual inspection.
4. Endpoints meeting all filtering criteria were selected for subsequent analysis, while those failing to meet any criteria were excluded.
5. Some endpoints were duplicated, those were removed.
6. The selected endpoints constituted the final dataset for generating visually appealing data schemas and conducting exploratory analyses.

Data Extraction:

1. Data extraction was performed using the OBIS-SchemaExtractor tool with the provided configuration.

```
2. correlationId: "8064359903421491904"
   endpointUrl: "https://lod.ehri-project-test.eu/sparql"
   graphName: null
   calculateSubClassRelations: true
   calculateMultipleInheritanceSuperclasses: true
   calculatePropertyPropertyRelations: false
   calculateSourceAndTargetPairs: false
   calculateDomainsAndRanges: true
   calculateImportanceIndexes: true
   calculateClosedClassSets: false
   calculateCardinalitiesMode: "propertyLevelOnly"
   calculateDataTypes: "propertyLevelOnly"
   sampleLimitForDataTypeCalculation: null
   sampleLimitForPropertyClassRelationCalculation: null
   sampleLimitForPropertyToPropertyRelationCalculation: null
   checkInstanceNamespaces: false
   minimalAnalyzedClassSize: 1
   principalClassificationProperties:
   - "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
   classificationPropertiesWithConnectionsOnly: []
   simpleClassificationProperties: []
   addIntersectionClasses: "auto"
```

```
exactCountCalculations: "no"
maxInstanceLimitForExactCount: 10000000
includedLabels: []
includedClasses: []
includedProperties: []
excludedNamespaces: []
predefinedNamespaces: null
enableLogging: true
```

3. Extraction success rate was reported as 44 out of 58 endpoints. Of the failed 14 data endpoints, 9 endpoints had a connectivity problem (class and property information can be retrieved via the web form only and not via an API call in the way, how it was tried) and 5 endpoints had performance problem (failure to retrieve classes or properties, or other difficulties within the extraction process).
4. Extraction time varied from 30 seconds to 5 hours and 24 minutes, just 3 out of 44 data sets had extraction time above 1 hour: 1h 14m, 2h 44m (Microsoft Academic Graph) and 5h 24m. depending on the complexity and size of the dataset. There have been 4 data sets with extraction time between 30 minutes and 1 hour.

Detailed account of the schema extraction and visualization process is available in the sparql_endpoints.xlsx file.

Result Format:

The result of the extraction was files in JSON format, containing structured data representing the extracted RDF schema information.

Data Import to PostgreSQL:

- All JSON files resulting from the extraction process were imported into a PostgreSQL server database.
- Data import was facilitated using the Data Shape Server tool.
- Some namespace adjustments were made post-import to ensure a more appropriate representation of namespaces in the data schema.

Post-Import Adjustments:

These adjustments aimed to enhance the clarity and coherence of the resulting data schema.

- Auto-generated namespace prefixes (where obtaining a “common-sense” prefix from *prefix.cc* was not possible) have been adjusted manually (by editing the *ns* table in the data schema in the DSS server).
- The presentation of the following schemas involved entity display name presentation adjustment in the DSS database:

- *databne*: import of class and property annotations from https://datos.bne.es/ (the annotations were not included in the SPARQL endpoint), followed by display name update for classes and properties from the English language annotations, and
- *bigcatbioinformatics*: the update of display name for classes and properties from the labels has been performed (the labels have been available in the SPARQL endpoint; the schema extraction needed to be repeated with marking *rdfs:label* as the property for entity labels).

Verification and Validation:

- Upon completion of the import process, the imported data was checked to ensure accuracy and completeness:
  - Typically, a cross-check with counting classes have been performed directly at the SPARQL endpoint;
  - the obtained data schema was checked to have both the class and the property information.

Data Visualization Procedure:

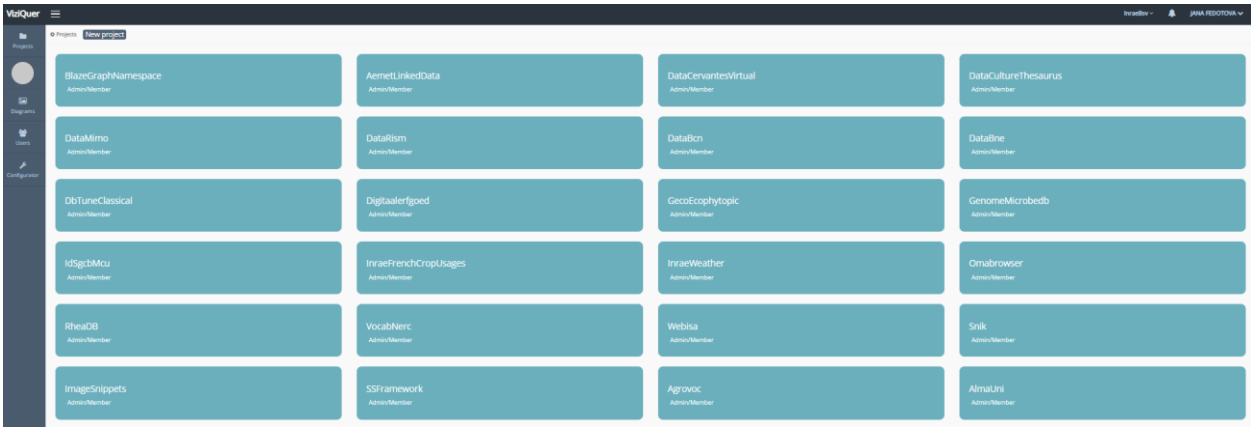
After importing the RDF schema data into PostgreSQL and preparing it for analysis, the experiment proceeded with data visualization using the ViziQuer application. The following steps outline the procedure for visualizing the data schemas:

Project Creation in ViziQuer:

- For each schema extracted and imported into the PostgreSQL database, a separate project was created in the ViziQuer application.
- The author initiated the project creation within ViziQuer, specifying the project name and relevant settings.

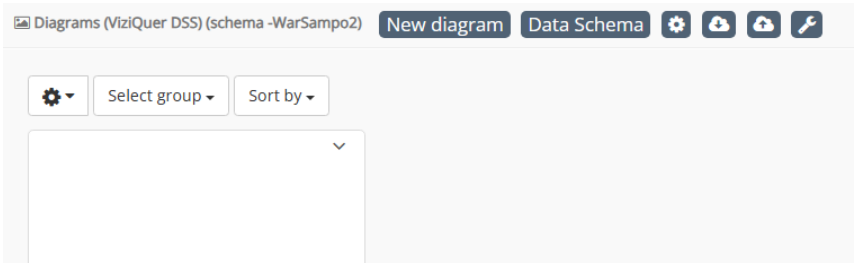
Opening the Project:

Once the project was created, it needed to be opened within the ViziQuer application.

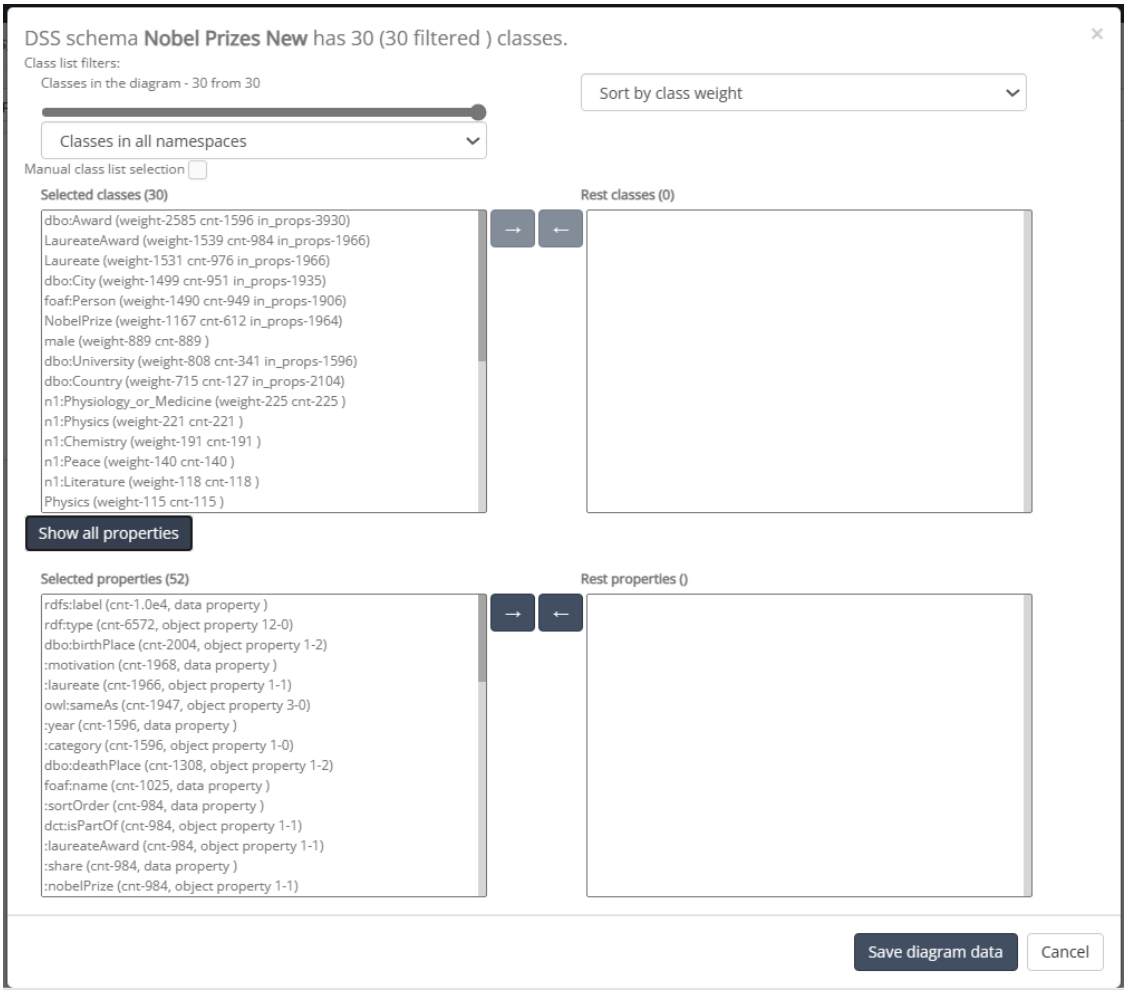


Configuring Visualization Parameters:

- After creating the new project, the author clicked on the 'Data Schema' button within *ViziQuer*.



- In the form, the author selected the parameters needed for visualization. These parameters could include properties list, class list, class list filters, etc.



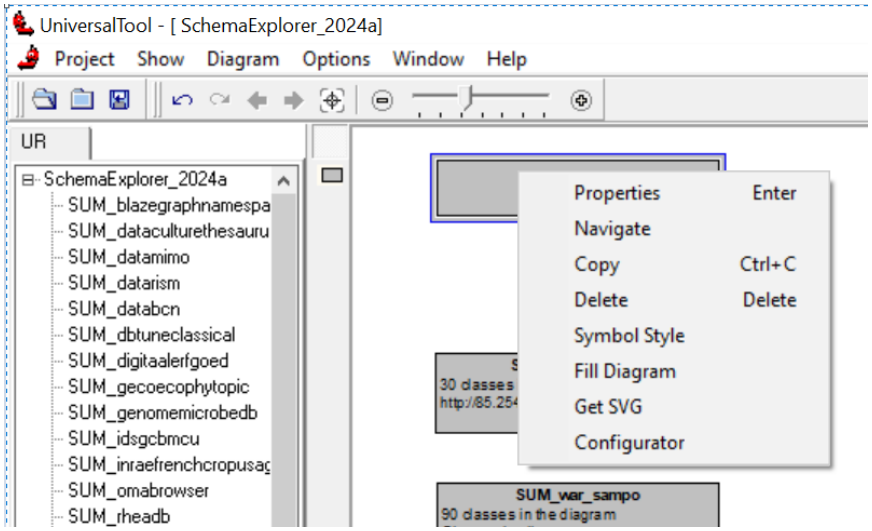
Saving Diagram Data:

- Once the necessary parameters were selected, the author clicked on the 'Save diagram data' button within *ViziQuer*.
- This action triggered the generation of JSON file with diagram data on the selected parameters.
- The downloaded JSON file contained structured data representing the visualized RDF schema.

Loading Diagram Data into Data Schema Explorer Tool:

- The downloaded JSON file was utilized to load diagram data into the Data Schema Explorer tool.
- The downloaded JSON file needs to be added into the Data Schema Explorer project used for the diagram visualization.

- Then for creating schema, new diagram seed symbol (represented by a grey square in the symbol palette) needs to be created. Next step is to right click on grey box and select ‘Fill Diagram’ option.
- To view new diagram double click on the diagram seed symbol (the grey box).



By following this procedure, the experiment leveraged the ViziQuer application to create visually engaging diagrams representing RDF schema data, which were subsequently loaded into the Data Schema Explorer tool for further exploration and analysis. This approach enabled researchers to gain insights into the semantic structures and relationships within the RDF data schemas, aiding in the interpretation and understanding of complex data models. This procedure was done for data that were extracted from endpoints.

Experiment results:

1. The OBIS-SchemaExtractor tool was employed to extract RDF schema data from the selected endpoints. The success rate was 75.8%.
2. The experiment resulted in the successful visualization and exploration of RDF schema data extracted from SPARQL endpoints. The obtained data schemas can be used for gaining the understanding about the respective SPARQL endpoint structure.
3. Overall, the experiment is deemed successful.