International Journal of Graph Computing
Vol. 1, No. 1 (2020) 39-69
© KS Press, Institute for Semantic Computing Foundation
DOI: 10.35708/GC1868-126723

A Framework for Exploration and Visualization of SPARQL Endpoint Information

Maria Krommydaand Verena Kantere

NationalTechnologicalUniversityof Athens,Athens,Greece
{mariakr,verena}@dblab.ece.ntua.gr

Received (11/12/2019)
Revised (01/29/2020)
Accepted (02/28/2020)

Abstract. Widely accepted standards, such as the Resource Description Framework, have provided unified ways for data provision aiming to facilitate the exchange of information between machines. This information became of interest to a wider audience due to its volume and variety but the available formats are posing significant challenges to users with limited knowledge of the Semantic Web. The SPARQL query language alleviates this barrier by facilitating the exploration of this information and many data providers have created dedicated SPARQL endpoints for their data. Many efforts have been dedicated to the development of systems that will provide access and support the exploration of these endpoints in a semantically correct and user friendly way. The main challenge of such approaches is the diversity of the information contained in the endpoints, which renders holistic or schema specific solutions obsolete. We present here an integrated platform that supports the users to the querying, exploration and visualization of information contained in SPARQL endpoints. The platform handles each query result independently based only on its characteristics, offering an endpoint and data schema agnostic solution. This is achieved through a Decision Support System, developed based on a knowledge base containing information experimentally collected from many endpoints, that allows us to provide case-specific visualization strategies for SPARQL query results based exclusively on features extracted from the result.

Keywords: SPARQL; Complex queries; endpoints; visualization; exploration; Decision Support Model.

1 Introduction

It was only twenty years ago when Tim Berners-Lee initiated the idea about the Semantic Web [33]. Its growth and expansion the following years was exponential as more and more data providers understood the importance of a common way to share information for their development and sustainability. A slow start, hindered by the need to manually annotate semantic data, was soon followed by a rapid acceleration when the process was automated. Encyclopedias, medical and scientific databases, large-scale projects such as Bio2RDF, the British Museum, the BBC Programmes and Music and the most prominent project Wikidata, the large knowledge graph of Wikipedia, were soon part of the Semantic Web and offered under the RDF format.

RDF is a data interchange model that aims to support the merging of information from heterogeneous sources with different schemas as well as the unobstructed update of schemas as needed without requiring any modification to the consumers of the information. RDF takes advantage of the structure of the Web and utilizes URIs to indicate relationships between entities. This data model provides the flexibility to merge structured and semi-structured schemas and share them in a uniform way. What makes the RDF model so widely used is that it is flexible and can be used to model information from heterogeneous sources. This is also what makes it a challenge to explore and understand, as data expressed in RDF is typically stored in large interconnected databases, without a homogeneous schema. Upon the increase of the volume of the information available on the Web, the need for a uniform way that facilitates the accessibility, the discovery and the understanding of the available information became prominent. Since being released as an official W3C recommendation in early 2008, SPARQL has evolved into the major query language for the Semantic Web and the RDF model. It has become the main standard for querying semantic data stores and is a key technology of the open data movement. SPARQL is supported by nearly all modern RDF based storage systems and is widely used in enterprise and public web contexts. There are two main challenges regarding the use of the SPARQL query language for the exploration of information. To begin with, novice users of SPARQL can easily perform simple and basic queries but require extensive training to utilize all the exploration capabilities of the language due to the wide range of functionalities. This is often deterring or disengaging for users that are not very familiar with the Semantic Web. In addition, the provided data rarely comply with strict models or have specific structure. Most users however are familiar with strict data models and relational databases as well as results that are complete and have specific structure. The diversity and flexibility of the SPARQL results is often hindering to the compilation of meaningful queries and the understanding of the information.

A SPARQL endpoint enables users, either humans or machines, to query a knowledge base using the SPARQL language. Results are typically returned in machine-processable formats, mainly SPARQL Query Results XML or JSON Format. Informally, a SPARQL endpoint is mostly conceived as a machine-friendly interface over a knowledge database. Accessing the information, forming the proper queries, understanding and representing the retrieved information is the responsibility of the user of the endpoint and not of the provider. As the number of the available endpoints increases, information for many different research areas is available, which is of interest to a wider audience, especially scientists and researchers, with limited knowledge of the Semantic Web.

To address the needs of such users a few endpoints were updated to offer access to the information through search engines and present the information in structured ways. These efforts however are limited to a very small number of the available endpoints and offer few, if any, exploration and visualization functionalities. The structured formats used, such as lists and tables, present the information in a human-readable format but without showing relations and connections between entities. Tools and applications that aim to support the querying, exploration and visualization of SPARQL endpoints are faced with many challenges. These include:

Querying support. Data in a SPARQL endpoints are fully accessible only for experts in the Semantic Web who are accustomed to forming complex SPARQL queries. Even then, navigating and exploring an unfamiliar SPARQL endpoint by hand can be quite laborious. For novice users only superficial exploration is possible without further support. Many tools, such as [29,72,66], are trying to provide the users with the needed support to explore and query the information, either by completely hiding the underlying SPARQL queries or offering graphical support to the compilation of the queries.

Content presentation. RDF is designed to facilitate machine interpretability of information and does not define a visual presentation model since human readability is not one of its stated goals. Presenting content primarily intended for machine consumption in a human readable and understandable way is very challenging. Most SPARQL endpoints provide the information in tables that vary on the number of columns based on the type of query. While such representations are intuitively close to what the average users, familiar with the relational schemas, expect, they are further from the nature of the RDF model, which is the graphical representation of information with connections between entities. Efforts to provide the information through graph-like visualizations however pose many challenges. Most approaches limit the usage to specific RDF vocabularies [49,60], SPARQL query types [67], domain-specific analysis [55] or already defined attributes [46].

Schema identification. The underlying data schema is not always available, or easily extracted. As a result, filtering and further querying is not always obvious or intuitive, based on the user expertise and the complexity of the schema next steps for the retrieval of the information may not be straightforward. Many tools [70,40,74] aim to extract the underlying schema, in order to support the exploration of the information. Such approaches however are not always successful as they fail for unstructured or overly complex schemas and only offer an overall estimation regarding the underlying schema.

Motivating Example. One of the most characteristic examples of information spaces that are challenging to explore and visualize is dictionary-like datasets. The information contained in them is incomplete, highly connected due to aliases and synonyms, domain-specific or represented in an unstructured way depending on the source of the information. Such datasets are also very valuable as they are utilized by universities, research institutes, public institutions and companies for knowledge organization. Also the format and characteristics of these datasets are important for research in information science as well as in the area of Linked Data and Semantic Web technologies.

Such an example of SPARQL endpoint that is very challenging to explore is the Standard-Thesaurus Wirtschaft (STW) Thesaurus for Economics [54]. This thesaurus provides 6.000 subject headings in two languages, English and German and it is considered the world's most comprehensive bilingual thesaurus for representing and searching for economics-related content. It utilizes more than 20,000 synonyms to cover not only all economics-related subject areas but also many related subject fields. The STW is published and continuously further developed by the Leibniz-Informationszentrum Wirtschaft [71], the German National Library of Economics, according to the latest changes in the economic terminology.

Due to the importance of the information that it represents the endpoint is of interest to a wide range of users with different needs. Economists and users less familiar with the Semantic Web focus on locating terms of interest, study their relationships with other terms and vocabularies and finding the translation of the terms between the two languages. Exploration systems should provide simple access to the information through keyword search, support the execution of exploratory queries through a user-friendly interface that requires from the user minimum input or knowledge of the Semantic Web, allow the retrieval of information related to one specific term based on its relationships with others and support the representation of the information in an intuitive way as graph.

Companies and institutions are more interested in exploring in-depth the dataset, discovering statistics regarding connections between synonyms from different subject fields and retrieving answers to complex queries. The users with experience of the Semantic Web should be able to write their own queries and get the proper visualizations.

Contributions. We present here a complete solution that supports both expert and novice users with querying SPARQL endpoints, exploring and visualizing the results in a dynamic way. Aiming to create a system that will overcome the challenges mentioned above we developed a client-server architecture that can offers:

Schema agnostic. Our system is carefully designed to handle any endpoint, without any information about the underlying schema, without any compromise on the user experience.

Decision Support System. We have developed a DSS that makes the decision regarding the visualization type that should be used for a query result. The DSS can provide specific parameters and layouts for queries that are to be visualized as graphs and specific charts for queries that contain less variables and aggregated information.

Knowledge database & Experimental analysis. We have accessed and analysed multiple SPARQL endpoints to collect information regarding possible query results. This information is used by the DSS to define the visualization parameters of the graphs. We have also performed a detailed analysis of the results, evaluating the range of parameters for the endpoints and associating them with the corresponding visualization parameters.

Query-specific visualization rules. We have studied the methodologies for choosing the right charts for the right data and we have utilized them to create proper rules that will match query types with the proper visualization chart. We provide case-specific visualizations for query results, based only on information and features extracted from them.

Integrated platform. In order to showcase the flexibility and the robustness of this approach, we developed an integrated platform that allows the users to query a SPARQL endpoint of their preference, either by writing their own query or by using a supportive form or by simply running a keyword search, and visualize the result based on its type and characteristics. Furthermore, the platform allows the user to filter the visualized information dynamically by exploiting the semantic annotations of the data and further explore the dataset by providing support and hints towards the next exploration steps.

The structure of this paper is as follows: In Section II, we present the system architecture for the exploration and visualization of SPARQL endpoint query results. In Section III, we present the Decision Support System. In Section IV we present the Integrated Platform. In Section V we present the review of the related work. Finally, in Section VI, we present the experimental analysis of SPARQL endpoints that was used for the creation of the knowledge database.

This paper is an extended version of the conference paper presented in Graph Computing 2019 [56].

2 System Architecture

We present the system architecture in Figure 1. The system can be divided into two main modules, the integrated platform and the Decision Support System (DSS). The integrated platform is the client component, accessible by the user and tasked with all the user-related interactions as well as the communication with the SPARQL endpoint while the DSS is the server component accessible through a dedicated interface. The integrated platform has three sub-modules, the *User Interface* which is responsible for visualizing the query results and supporting the user is locating and exploring the information of interest through a series of functionalities, the *Query processor* which receives the SPARQL query and queries in real time the selected endpoint and the *Feature Extractor* which extracts the needed features from the result and forwards this information to the DDS interface. The Decision Support System has three key sub-modules, the *Knowledge Database* that contains the raw data needed to make a decision, the *Decision Model* that has all the rules and logic of the decision making and the *DSS Interface* that receives the extracted features from the integrated platform and returns the decision of the DDS in the form of visualization parameters. We present below the modules of the architecture in detail.

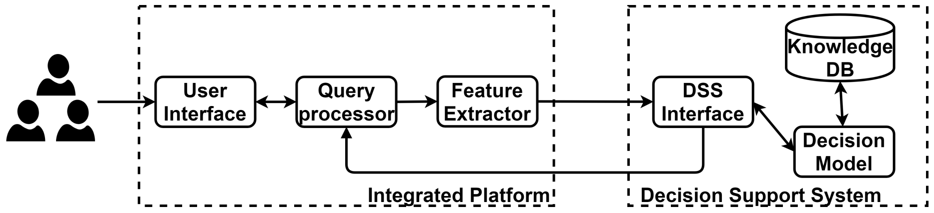


Fig. 1. System Architecture

3 Decision Support System

A DSS is an information system that supports decision-making activities. DSSs are designed to support the operational planning and help people make decisions about problems. Their main contribution is that they can support problems that are rapidly changing and not easily specified in advance. A DSS can be built in any knowledge domain as long as enough information can be collected to support the decision model. A DSS is designed to combine relevant information provided from a knowledge base and models to solve problems and make decisions. There are three key components to a DSS architecture. The knowledge database that should contain data presenting the real word and serve as the basis for the system. The model, the core logic of the system where based on the available information all the decisions are made and the interface where the current problem/situation is given as an input and a decision is returned as an output.

We believe that a DSS can be very useful in the context of the visualization of SPARQL query results. To begin with, the problem of effectively visualizing a specific query result is within the core problems that the DSSs can handle. This is due to the fact that it is a problem that cannot be specified in advance. Two query results are expected to present high diversity regarding their characteristics even for a single endpoint. A DSS provides the flexibility, through the modeling process, to have the needed rules to support such diversity.

Example 1. An indicative example for the STW Thesaurus for Economics can be the diversity between a query from a data scientist interested in the number of appearances of each predicates available in the dataset and a query from an economist interested in retrieving all the terms containing the keyword *Economy* and their description. In the first case, the result is showing the distribution of the 100% of the predicates and should be represented as a pie chart while in the second case the result is 121 triples containing terms and their description which should be represented as a graph.

In addition, modeling data to visualization types using the proper parameters is an intuitive process that follows specific empirical rules when the characteristics and format of the data are known. Last but not least, dynamically offering case-specific visualization parameters for each query result allows the exploration of any SPARQL endpoint without requiring any knowledge for the underlying schema or enforcing any limitation to the exploration.

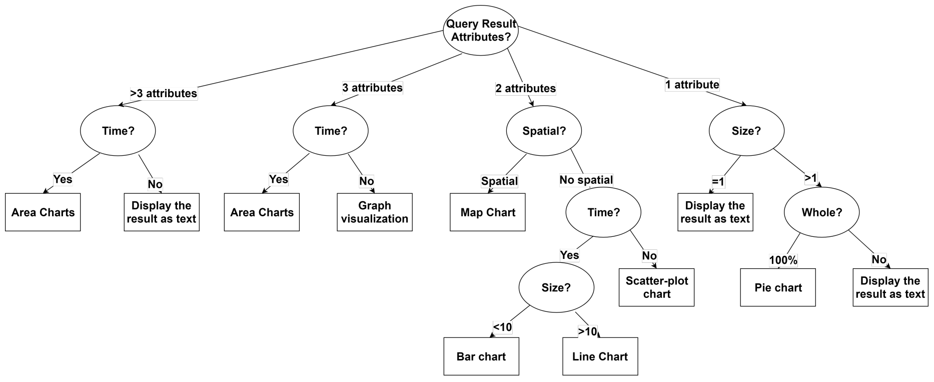


Fig. 2. The tree showing the decision making process for the model

We have developed a decision model that proposes two visualization categories, graphs for queries results containing triplets of information without a time variable and charts for query results containing one or two variables and aggregation functions. In order to determine the parameters that are needed for the graph layout of a query result we have developed a knowledge database with information from many available endpoints. The knowledge database allows us to determine if a graph is highly connected, if it contains a lot of descriptive information or follows a specific pattern allowing as to choose the right layout algorithm and parameters. Regarding the choice of the appropriate chart we have created a series of rules as part of the decision model that interpreter known data visualization rules to specific query types. We present below the implementation details of the three sub-modules of the DSS.

3.1 Knowledge Database

According to studies that were carried out on logs of endpoints, aiming to study patterns in the queries and the users of the endpoints [34,59,64], the majority of the queries are *SELECT* queries for triples without aggregation. This means that the most important part of the visualization that we need to properly parameterize are the graphs. In order to achieve that we need to know what are the expected range for the characteristics of such queries. We have accessed and analyzed multiple SPARQL endpoints to collect information regarding the result size limit, the total number of unique predicates, the total number of unique entities, the most/least connected entities, the most/least used predicate, the min/max string length for entities and predicates and the average node degree. We present in Section 6 the details regarding the experimental methodology and results. The collected information allows us to determine if the characteristics of a result are within the expected limits, identify potential issues and choose the right visualization parameters. The information collected is stored in a relational database accessible by the decision model.

3.2 Decision model

As presented in Figure 2, we have developed a deterministic decision tree that identifies the proper visualization type based on the query type and result characteristics. Triplets of information containing no time information are visualized as graphs. In Section 3.2, we present which features of the query result are examined and how they are utilized in defining the visualization parameters for the graphs. Here, the information available in the knowledge database is used to support the process and provide the needed information regarding expected values. In Section 3.2, we present the decision making process regarding the selection of the right chart for query results with one, two variables or three variable, one of them time related.

Graphs Graphs are used for all the query results that are in the form of triplets and do not contain any time information. The presence of time is evaluated based on the OWL-Time [25] ontology of temporal concepts which has been created for describing the temporal properties of resources and is associated with the *http://www.w3.org/2006/time#* namespace.

Graphs are dependent on many visualization parameters that can be adjusted to accommodate a wide range of query results including the layout algorithm that can be exploited to support the exploration of the results. Specific characteristics of the data to be visualized, such as the node size, node degree and the overall size directly affect choices regarding the overlapping percentage, the compactness and the edge length of the visualization. In order to specify the graph visualization parameters, the features of the query result are examined and evaluated based on the information available at the knowledge database. The evaluation method is presented here:

- Node size. In order to determine the size of the nodes, the average string length of the labels available in the query result are examined. The value is then compared with the information available at the knowledge database, the minimum and maximum values of the label lengths of the endpoint. Based on this comparison, the node size is determined.
- Node degree. Aiming to determine the density of the query result, the average node degree is calculated. The value is then compared with the information available at the knowledge database, the average value of the node degree for the endpoint, and the result density is determined.
- Result size. The result size is compared to the maximum result size that the endpoint is support. In the case that this is reached, an additional parameter is added to the provided response, to ensure that the user is notified accordingly.
- Edge length. Nodes that do not contain long textual descriptions are relatively small in size and can be brought closer in the two-dimensional space. The maximum edge length is proportional to the node size.
- Node overlaps. The selected node size determines the allowed percentage of overlapping between nodes. As a rule of thumb, for larger nodes a higher degree of overlapping is acceptable given that the user gets an understanding

of the information provided even without reading the complete information. Based on this rule the overlapping percentage is proportional of the node size.
– Edge crossing. Minimizing the number of edge crossings is very important for the readability of the graph, the path navigation and the node exploration. To accommodate that, special graph layouts are investigated and hierarchical and tree-like graph layouts are prioritized. Also, the default graph layout algorithm used if the query result does not comply to any specific structure, which is described in detail below, is designed to minimize the edge crossing.
– Graph area. Minimizing the overall graph area is not considered priority for the visualization. This is mainly due to the fact that the user interface offers many functionalities that support the exploration and navigation of the information, ensuring a user-friendly experience even when the graph is more than a few screens in dimensions.
– Node distribution. Uniform spatial distribution of the nodes ensures that the graph is easily explored and navigated. To this end, the minimum area that the graph covers is calculated proportionally to the result and node size.

In the cases where the decision model has to provide visualization parameters for a SPARQL endpoint that has not been examined before, meaning that it is not included in the knowledge database, then the average values of all the endpoints included in the knowledge databases are used instead. The url of the endpoint is saved on the server and the endpoint is examined offline and included in the knowledge database if possible.

As already discussed, identifying the most suitable graph layout algorithm for the query result is key for the proper exploration of the information. Tree, hierarchical, star and circuit graph structures are first examined and applied if possible, in any other case the more generic solution of the force-directed layout algorithm is chosen. Also, for query result that contain symmetries, it is very important to respect and visualize them to ensure the comprehension of the visualized information. In details:

– Tree or Hierarchical structure. In order to check if a directed graph, like a query result, is a tree we need to examine all triplets of information as follows. The first step is to locate the root node of the tree, a vertex with only outgoing edges. In the cases where there is more than one vertex with only outgoing edges or there is no such vertex then the triplets do not comply with the tree structure. After locating the root node we do a Depth First Search starting from it, if the search encounters the same vertex twice, indicating that it can be reached from two different paths, then the result does not comply with the tree structure. If the search concludes with unexplored vertices, then the graph is not connected and cannot therefore be a tree. If the Depth First Search contains all the nodes only once then the triplets comply with the tree structure and can be presented as such.
– Star layout. In order to identify a result that complies with the star layout we examine the node degrees. If the minimum node degree equals 1, the maximum node degree equals with $resultsize - 1$ and the average node

degree equals $2 * (resultsize - 1)/resultsize$ then the query result has one central node and is presented using the star layout.

– Circuit layout. Similarly, the query result can be represented using the circuit layout when the minimum, maximum and average node degree equals with 2.
– Planar graph. Planar are called graphs that can be presented in the two-dimensional space without any edge crossing. We are not examining the possibility of offering the query result using a planar visualization, taking into account that there are many discussions about whether they offer any visual improvement over the non-planar visualization and the cost of computing if a graph is planar or not.
– Force-directed graph layout. For query results that does not comply with any of the above mentioned structures, we employ a generic yet robust algorithm for the graph layout. Force-directed graph drawing algorithms position the nodes of a graph in two-dimensional space trying to minimize edge length and edge crossing. The idea is to assign forces among the edges and the nodes and use them to minimize their energy. On one hand, spring-like attractive forces based on Hooke's law are used between nodes that are connected with an edge to bring the pair closer in space. On the other hand, repulsive forces like those of electrically charged particles are used between all pairs of nodes, no matter if they are connected or not. The balanced state of these forces, ensure that the edges have uniform lengths and nodes that are not connected are further apart in space.
  Such algorithms have many benefits, as a result of the force balancing that ensures specific features. To begin with, the result is of high quality, offering uniform edge length, enforcing the spatial distribution of the nodes and emphasizing symmetries in the data. Also, it provides the needed flexibility as the balanced state of the forces can easily parameterized, as an example increasing the attractive forces brings the nodes closer in the two-dimensional space, based on the specific characteristics of the query result. Finally, the algorithm mimics the physical world providing an output that can be intuitively understood and accepted.

For example, for query results where the average string length is near the maximum that we have encountered the potential issue it that few nodes containing descriptive information will affect the overall dimensions of the nodes, expanding the space occupied by the graph and making its exploration a challenge. So, in such cases, the allowed overlapping percentage is 30%, the maximum allowed one and the node size is 150px, again the maximum one. This allows us to show to the user the majority of the information in the nodes, but also keep the overall size of the graph relatively small to allow the exploration of the connections between nodes. Similarly, the query result for all the synonyms of the term *Author* has result size four and a max node degree also four. This means that this query result should be visualized using the star layout algorithm as shown in Figure 3. In Figure 4 we present the exploratory query result for the term *http://www.w3.org/2004/02/skos/core#Collection*, which has been visual-

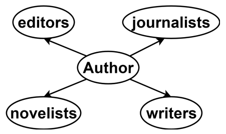


Fig. 3. Star graphical representation of the synonyms for the word 'Author'

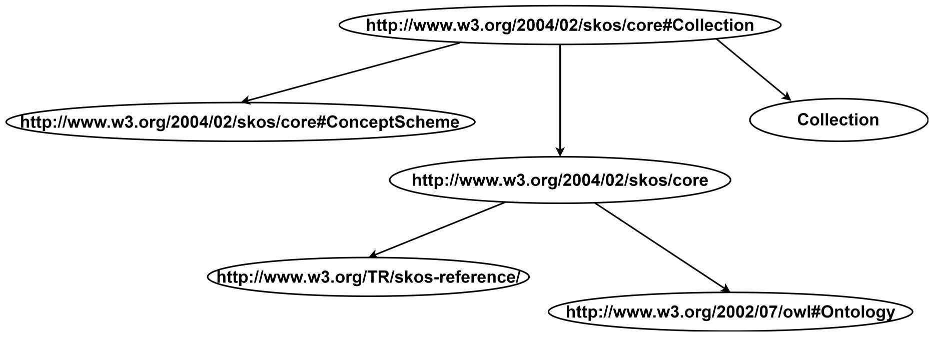
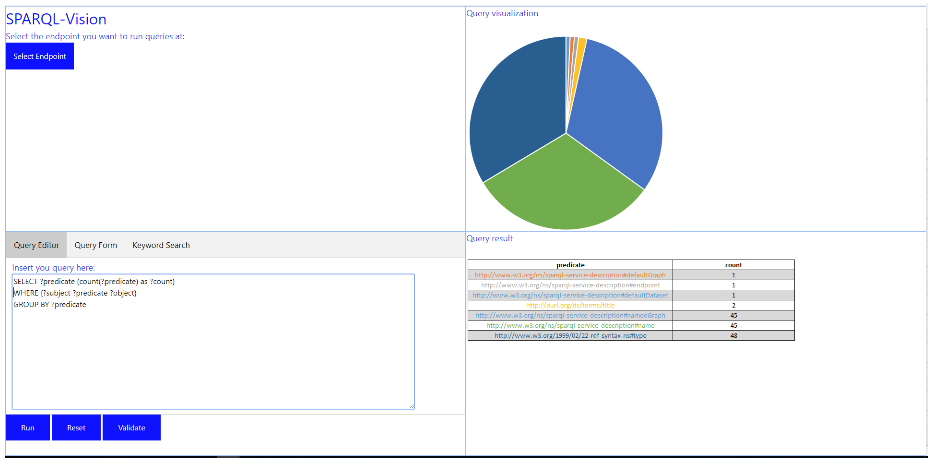

http://www.w3.org/2004/02/skos/core#Collection
http://www.w3.org/2004/02/skos/core#ConceptScheme
Collection
http://www.w3.org/2004/02/skos/core
http://www.w3.org/TR/skos-reference/
http://www.w3.org/2002/07/owl#Ontology

Fig. 4. Tree-like representation of an exploration query

ized using a tree-like approach due to the fact that there was one node with no incoming edges and no closed paths.

Charts Choosing the right chart for the right data is not a scientifically defined task, it is based on intuition and takes into account the purpose of the visualization and the audience that the visualization is addressing. Aiming to provided a widely accepted set of rules, there are many studies that performed experiments aiming to identify the most suitable visualization type to be used based on the data format and content [68,73,41,53,57]. These studies have formed a set of empirical data visualization guidelines for choosing the right chart for the right data that are widely accepted. We have adopted these guidelines and we have extended them and matched them with query types in order to create specific decision rules for the model. We present here the defined rules, giving an intuitive description of the data visualization guideline, emphasizing the features of the query results that are important for each rule and concluding with the query types associated with each visualization type.

Pie chart. We present first the pie chart as it is the only chart available that can represent only one variable. Specifically, it is used to represent the distribution of the 100% of a value, when there is no time element and there are at most 10 parts. If there are too many parts, or the percentage distribution is unbalanced, having one category over 90% and many others sharing the remaining 10%, then the visualization is not aesthetically appealing it is not displayed to the user. This visualization type is ideal for queries that have an aggregation with a group by clause over a variable without any filtering, such as a having clause. The visualization may include, based on the user selection, also the labels of the categories represented by the percentages. This additional information is included in the textual representation where tuples are color matched with the chart.



```
SELECT ?predicate (count(?predicate) as ?count)
WHERE {?subject ?predicate ?object}
GROUP BY ?predicate
```

predicate	count
http://www.w3.org/ns/sparql-service-description#defaultGraph	1
http://www.w3.org/ns/sparql-service-description#endpoint	1
http://www.w3.org/ns/sparql-service-description#defaultDataset	1
http://purl.org/dc/terms/title	2
http://www.w3.org/ns/sparql-service-description#namedGraph	45
http://www.w3.org/ns/sparql-service-description#name	45
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	48

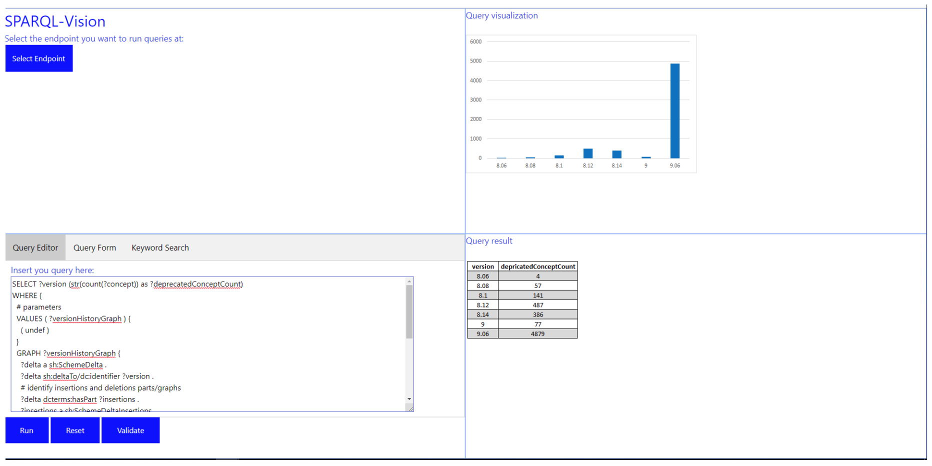
Fig. 5. Count the times each predicate appears

Example 2. As an example, a user interested in the times each predicate appears at the STW Thesaurus dataset would expect the result to be visualized as a pie chart. This is due to the fact that the query result represents the distribution of all the predicates in the dataset, there is a limited number of predicates to be visualized, only seven, provided by a group by query with an aggregation without any filtering. The visualization of this query result is shown in Figure 5, where the user has also chosen for the predicate labels to be included in the result.

Bar charts. Bar charts serve two purposes, the monitoring of one variable over the course of time or the quantification of a value related with a classification. Bar charts are preferred for small datasets, at most with 12 parts, like the monthly total value on the income of a company. The decision model chooses this chart when the extracted features indicate that the query result has only two variables, one of them is a numerical variable and the other is either time-related or descriptive, and the size of the result is less than 12. In contrast with pie charts, bar charts are not aware of the total distribution of the variable that they represent or if it an aggregated value. As an example, a bar chart can visualize the average age of retirement for people in ten most stressful work fields.

Example 3. Taking advantage of the *skos-history version store* which has been created with different versions of STW Thesaurus for Economics we can create useful queries that show the changes between versions over time. As an example we can retrieve the count of deprecated concepts between versions. The result of this query has two variables, one is time-related and a size of 7, so it is represented as a bar chart as shown in Figure 6.

Line charts. Line charts are complimentary to the bar charts, as they are used to monitor the same datasets but when the size is more than 12. In detail,



```
SELECT ?version (str(count(?concept)) as ?deprecatedConceptCount)
WHERE {
 # parameters
 VALUES ( ?versionHistoryGraph ) {
  ( undef )
 }
 GRAPH ?versionHistoryGraph {
  ?delta a sh:SchemeDelta .
  ?delta sh:deltaTo/dc:identifier ?version .
  # identify insertions and deletions parts/graphs
  ?delta dcterms:hasPart ?insertions .
```

version	depricatedConceptCount
8.06	4
8.08	57
8.1	141
8.12	487
8.14	386
9	77
9.06	4879

Fig. 6. Count of deprecated concepts between STW Thesaurus versions

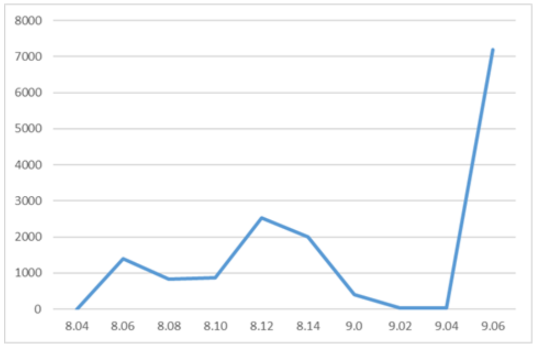


Fig. 7. Count of new concepts inserted per STW Thesaurus version

line charts show the variation of one variable with a lot of values over the course of time or the quantification of a value when classified over multiple classes. This visualization type is chosen when the query result has two variables with one of them numerical and the other either time related or descriptive and size more than 12.

Example 4. Taking again advantage of the *skos-history version store* we can retrieve the count of new concepts inserted per version. The result of this query has two variables, one is time-related and a size of 10, so it is represented as a line charts as shown in Figure 7.

Scatter-plot charts. We have discussed the cases of datasets with two variables that include either a time element or descriptive information, we present here scatter-plot charts which are used for dataset that have two numerical values. They represent the information as spots over a grid-like area, where each axis can have a different scale to accommodate the minimum and maximum values of the representing variable. This chart does not have a size limit given that its overall dimensions are proportionally larger to the size of each visualized part. This visualization type is used for query results that contain exactly two numerical variables without a time element.

Map charts. Map charts are used to represent time invariable spatial information. This visualization type is used for datasets that contain spatial information, either coordinates or region names. The displayed information may be a numeric value, then a one-color visualization with gradient is used as the quantitative indicator, or descriptive where a multi-color schema for the categories is adopted. This visualization type is used for query results that contain spatial information. For uniformity reasons we categorize these queries as two-variable ones, one variable for representing the displayed information and one that provides the spatial element. Often, however, the spatial information may be available as two variable, latitude and longitude in the result. In such cases we merge the two variable in one pair of coordinates to facilitate their further processing.

Area charts. Area charts are used to represent how parts of a whole change over time. This visualization type is ideal for datasets that contain at least three variables, one of them is time related. The chart shows the variation of the variables and their comparison in each time stamp. It can be presented in two versions, the stacked area chart where the distribution of the variables is not examined and the 100% area chart where the variables represent the distribution of the 100% of a value at any given time. This chart is used for queries that have an aggregation along a group by clause over two variables, represented over time. Here, the presence of filtering functions is examined to ensure the use of the proper area chart type.

Descriptive representation. The chart types presented above do not cover all the possible cases for query results. In the case of a result with only one variable a chart is created only when the query contains a group by clause with any filtering using a pie chart. In all the other cases the available information is not sufficient to create any visualization. Also, in the case where the query result has more than three attributes without any time element, the information is displayed in a structured table and no visualization is provided, given that the dataset in lacking the needed cohesion.

We have implemented the decision model as a Java application with strict deterministic rules. The extracted features are mapped to the rules and the most fitting visualization type is selected.

3.3 DSS interface

We have implemented the DSS interface as a standalone Tomcat application that receives as input a JSON data object that contains the extracted features regarding the query result. The features that are included in the data object are such to support the choice of the visualization type independently of the content of the query, the endpoint queried or the underlying schema. The features included in the JSON object are:

– Number & type of variables: list of variable types

- Time element: boolean
- Spatial element: boolean
- Result size: numerical
- Aggregation function: boolean
- Group by function: boolean
- Filtering function: boolean
- Average string length: numerical
- Min/Max/Average node degree: numerical
- Root node: boolean
- Depth First Search: list of nodes

4 Integrated Platform

4.1 User Interface

A key component of the integrated platform is the user interface. This component is responsible for all the interactions with the user and allows the user to navigate and explore the information in an efficient and semantically meaningful way. It supports the composition of queries, ensures the validity of the user queries, presents to the user the retrieved information in dynamic and case-specific visualizations and allows the exploration of the information through filtering and isolation techniques. To achieve all the above a series of functionalities are provided through the user interface.

Query composition. The user interface offers two ways for a user to write a SPARQL query. The first one is addressed to expert users, a single text box allows the user to type in any query. The second way is addressed to less experienced users that have only some basic knowledge and understanding of the SPARQL language. Here, the user is guided through the query composition by a form that has drop-down menus when applicable and requires minimal free text input from the user.

Query validator. To further support the user in composing their queries we have incorporated to the interface a query validator. After typing the query, the user can ask to validate the query and any issues are highlighted and the proper message with suggestions is shown. Even if the user chooses to submit the query without validating it, the validation process is always triggered before sending the query to the endpoint to avoid any invalid responses and error messages.

Overview of the query result. We present a text overview of the result to help the user get acquainted with it before exploring it. The textual representation of the query result is also used as a legend for the charts in case it is required by their type. This way the user can easily understand the displayed information by color matching the relevant text to the chart part. Furthermore, for the graph visualization, an additional overview is produced by isolating the most connected nodes, based on their node degrees, and visualizing them with respect to the spatial placement in the graph. The overview is interactive, meaning that the user can click on an area of the overview and the visualization panel is centered to the corresponding part of the graph.

Dynamic visualization. For large query results, or charts that are contain a lot of information such as scatter-plots, the mapping of the query result to the visualization part might not be obvious. For this reason, the query result visualization is not presented as a static image but as a dynamic object, responsive to user actions. This allows the user to select a part of the visualization and see highlighted the corresponding result part or chose a result part and see how it is depicted in the visualization.

Customizable filtering functions. As it will be presented in Section 6, only one in three endpoints enforces a query size limit and even in such cases it can be as high as 100000. Recognizing that navigating through such large query results can be ineffective and disengaging, the system provides to the user a series of filterng capabilities that can be used to restricted the displayed information. The interface enables the user to use one or multiple filtering criteria at the same time. A very characteristic example where this functionality is of great applicability is when the query contains multiple categories of semantic information. For example, when querying information about a specific terminology the result may contain information in different languages as well as synonyms and antonyms. A user can apply the filtering functions provided to display only results in English and terms that are synonyms to the original term. A different example would be a user visualizing aggregated the average expenses of a company per month. In this case, there might be a need to exclude from the visualization months before a limit. It is worth noting that all the provided filtering functions can also be achieved through an updated SPARQL query. However, by filtering the already retrieved information at the interface we allow the user to explore different filtering scenario in an interactive way, with a short response time and avoid over-querying the SPARQL endpoint.

Keyword search. Another way designed to support the novice user in getting started with the exploration of a SPARQL endpoint and locate information of interest is through keyword search. The term is queried against the selected endpoint and the query results are presented to the user following the same data flow as any other result set.

Path navigation. For results that are visualized as graphs, the user is able to choose one node and follow all the paths of the graph that include this node, while the rest of the information is hidden. This way, irrelevant information is eliminated allowing the user to focus exclusively on the node and paths originating on it that are of interest. In contrast with most approaches in which a path can be traversed on one direction only, in our system when a node is chosen, all neighbors, either incoming or outgoing are visualized.

Sub-result isolation. The user can also choose to isolate a part of the result query, to better exploit and navigate the information that is present there. In order to make the isolation of the information user-friendly, the information to be isolated is chosen either by selecting the relevant part of the visualization or the text from the result overview.

Exploration support. Aiming to further support the novice user with the exploration of the available information the visualized result is interactive and upon user actions suggests further queries. An indicative example is when a user selects one specific result entity, then queries that retrieve its neighbors are suggested.

4.2 Query processor

When a new query is submitted by the user a process is initiated by the integrated platform. First the query is validated, if any syntactic issues are identified, the user in notified and the query is not send to the endpoint until the user has corrected it. The valid queries are asked in real time to the selected endpoint. This approach is selected as it gives the integrated platform the flexibility to connect with any endpoint and provide to the user real time information based on the latest update. The alternative would be to duplicate the information of all the endpoints users are interested in to a dedicated back-end. This approach has main drawbacks as it is an expensive, time consuming process that would incapacitate the role of the SPARQL endpoints to provide up-to-day data. Also it would have a huge overhead to the back-end and threaten the sustainability of the platform as there would be need for storage of Terabytes of data and would delay the provision of information to the user in case he chose to visit an endpoint never before encountered by the platform. While temporary unavailability of the endpoint or poor performance may affect the user experience this is evaluated as a rarer and minimal discomfort to the user when compared with ensuring the flexibility and sustainability of our approach. The query result is then forwarded to the feature extractor.

4.3 Feature Extractor

The Extractor receives the query and the query result and extracts all the information needed for the DSS. This is properly formatted in a custom JSON object and forwarded to the DSS interface.

5 Related work

Visualisation tools The Tabulator [32] project is an attempt to demonstrate and utilize the power of linked RDF data with a user-friendly Semantic Web browser that is able to recognize and follow RDF links to other RDF resources based on the user's exploration and analysis. It is a generic browser for linked data on the web without the expectation of providing as intuitive an interface as a domain-specific application but aiming to provide the sort of common user interface tools used in such applications, and to allow domain-specific functionality to be loaded transparently from the web and be instantly applicable to any new domain of information. The Linked Data Query Wizard [50] is a web-based tool for displaying, accessing, filtering, exploring, and navigating Linked Data stored in SPARQL endpoints. The main innovation of the interface is that it turns the graph structure of Linked Data into a tabular interface and provides easy-to-use interaction possibilities by using metaphors and techniques from current search engines and spreadsheet applications that regular web users are already familiar with. Linked Data Visualization Model (LDVM) [36,37] allows to dynamically connect data with visualizations. In order to achieve such flexibility and a high degree of automation the LDVM is based on a visualization workflow incorporating analytical extraction and visual abstraction steps. Each of the visualization workflow steps comprises a number of transformation operators, which can be defined in a declarative way. As a result, the LDVM balances between flexibility of visualization options and efficiency of implementation or configuration. This has been expanded [49], to support the visualization of data provided using the RDF Data Cube Vocabulary.

Visualbox [42] is a system that makes it easier for non-programmers to create web visualizations based on Linked Data. Visualbox provides a unified environment that supports the whole process of creating a visualization based on a SPARQL query. It runs a query on the server and provides a useful caching mechanism that allow users to visualize the data even if an endpoint is down or unresponsive. gFacet [48] is a browsing approach that supports the exploration of the Web of data by combining graph-based visualization with faceted filtering functionalities. The graph-based visualization facilitates a comprehensible integration of different domains; the use of facets supports a controlled filtering of information. With gFacet, users are enabled to browse the Web of data efficiently and to retrieve information from different user-defined perspectives. Sgvizler [67] is a small JavaScript wrapper for visualization of SPARQL results sets. It integrates well with HTML web pages by letting the user specify SPARQL SELECT queries directly into designated HTML elements, which are rendered to contain the specified visualization type on page load or on function call. Sgvizler supports a vast number of visualization types, most notably all of the major charts available in the Google Chart Tools, but also by allowing users to easily modify and extend the set of rendering functions.

Facet Graphs [46] allows humans to access information contained in the Semantic Web according to its semantics and thus to leverage the specific characteristic of this Web. To avoid the ambiguity of natural language queries, users only select already defined attributes organized in facets to build their search queries. The facets are represented as nodes in a graph visualization and can be interactively added and removed by the users in order to produce individual search interfaces. This provides the possibility to generate interfaces in arbitrary complexities and access arbitrary domains. Explorator [38] is an open-source exploratory search tool for RDF graphs, implemented in a direct manipulation interface metaphor. It implements a custom model of operations, and also provides a Query-by-example interface. Additionally, it provides faceted navigation over any set obtained during the operations in the model that are exposed in the interface. It can be used to explore both a SPARQL endpoint as well as an RDF graph in the same way as “traditional" RDF browsers.

CubeViz [60] is a flexible exploration and visualization platform for statistical data represented adhering to the RDF Data Cube vocabulary. If statistical data is provided adhering to the Data Cube vocabulary, CubeViz exhibits a faceted browsing widget allowing to interactively filter observations to be visualized in charts. Based on the selected structural part, CubeViz offers suitable chart types and options for configuring the visualization by users. By employing advanced introspection, analysis and visualization bootstrapping techniques CubeViz hides the schema complexity of the encoded data in order to support a user-friendly exploration experience. Timely YAGO [69] enhances facts extracted from Wikipedia with temporal validity information. Temporal facts are extracted from Wikipedia infoboxes and lists in articles. These facts serve as the backbone of a temporal ontology and are used for bootstrapping the extraction of temporal facts from free text in our ongoing project. Payola [55] is a framework for Linked Data analysis and visualization. It provides end users with a tool enabling them to analyze Linked Data in a user-friendly way and without knowledge of SPARQL query language, through domain-specific analysis and reusable visualization plugins. D3SPARQL [52] is an open source library which can be embedded in any Web page, performs a SPARQL query via AJAX call, transforms the result and visualizes data with the help of D3.js library.

PowerAqua [58] is an ontology-based Question Answering system that is able to answer queries by locating and integrating information, which can be massively distributed across heterogeneous semantic resources. It also performs a deep exploitation of the available semantic information, providing query disambiguation, as well as knowledge fusion and ranking mechanisms to successfully elicit the most accurate answers to user queries. RelFinder [47] is an approach that automatically reveals relationships between two known objects and displays them as a graph. Since the graph that visualizes the relationships can still become large, interactive features and filtering options were added to the user interface that enable a reduction of displayed nodes and facilitate understanding. SPARQL-visualizer [35] aims to facilitate the design process of a shared ontology, where domain experts, software developers and ontology engineers collaborate. As they typically have a different view on the ontology and understanding of the technology, it can be difficult to communicate proposals within the group. Sample data, queries and results of them are visualized in table or graph form to support their collaboration. Haystack [51] is a platform for creating, organizing and visualizing information using RDF. It is based on the idea that aggregating various types of users' data together in a homogeneous representation, agents can make more informed deductions in automating tasks for users. Sextant [62] is a Web-based system for the visualization and exploration of time-evolving linked geospatial data and the creation, sharing, and collaborative editing of "temporally-enriched" thematic maps which are produced by combining different sources of such data.

Query writers Konduit VQB [29] provides a way for users to build SPARQL queries in an intuitive way, with having no or little knowledge about the querying language. This does not mean a complete abstraction from the underlying details, but provides an interface that suits the needs of both novice and expert users. QueryVOWL [45] is an approach for visual querying that reuses graphical elements from the Visual Notation for OWL Ontologies and defines SPARQL mappings for them. The goal is a visual query language that is intuitive and easy to use, while remaining flexible and preserving most of the expressiveness of SPARQL. SPARQL Builder [72] is an intelligent tool by which users with no knowledge of SPARQL can generate SPARQL queries and retrieve results satisfying their requirements. SPARQL Builder collaborates with TogoTable, a web application enabling biological researchers to upload their data in a table form and add annotations obtained from SPARQL endpoints. The goal of the tool is to support users in constructing SPARQL queries using TogoTable. NITELIGHT [66,65] is a graphical tool for semantic query construction that is based on the SPARQL query language specification. The tool supports end users by providing a set of graphical notations that represent semantic query language constructs. The tool also provides an interactive graphical editing environment that combines ontology navigation capabilities with graphical query visualization techniques. Paged Graph Visualization [39] is a new semi-autonomous tool for RDF data exploration and visualization. It consists of two main components, the explorer and the pager, a high performance main-memory RDF storage system. Its main strategy is to begin with a small graph and provide the tools to incrementally explore and visualize relevant data of very large RDF ontologies. MashQL [61] is a query-by-diagram language that regards the Internet as a database and generalizes the idea of mashups. People are allowed to build data mashups diagrammatically. MashQL queries are translated into and executed as SPARQL queries. The novelty of MashQL is that it allows querying a data source without any prior understanding of the schema or the structure of this source. SparqlFilterFlow [44,43] is an approach for visual SPARQL querying based on the concept of extended filter and flow graphs. In contrast to popular approaches, the queries can be created entirely with graphical elements. SparqlFilterFlow considers most features of SPARQL and can hence also be used for the construction of complex query expressions. SMART [31], Semantic web information Management with automated Reasoning Tool, is an open-source project, which aims to provide intuitive tools for life scientists to represent, integrate, manage and query heterogeneous and distributed biological knowledge. Features include semantic query composition and validation, a graphical representation of the query, and the retrieval of pre-computed inferences from an RDF triple store.

Schema extraction TBox visualization [70] aims to extract and visualize the information on the used schema, also called TBox from SPARQL endpoints. Rather than relying on given TBox information, the tool infers what a TBox for the available ABox data could reasonably look like based on several SPARQL queries. This information is incrementally added to an interactive graph visualization based upon the Visual Notation for OWL Ontologies. A node-link-based graph visualization is chosen, as it allows users to grasp certain structural criteria at a single glance, such as the presence of highly linked central classes or largely disjoint clusters of classes, before proceeding to a deeper analysis. ViziQuer [74] asks the user to provide an address of a SPARQL endpoint that is of interest, then it extracts and visualizes graphically the data schema of the endpoint. The user is able to overview the data schema and use it to construct a SPARQL query according to the data schema. The tool extracts a simplified data schema by using a predefined sequence of SPARQL queries at the SPARQL endpoint. This process can take a while since schema retrieval depends on ontology size and speed of the SPARQL endpoint while only typed data are supported. Afterburner [40] implements an analytical RDBMS in pure JavaScript so that it runs completely inside a browser with no external dependencies. It generates compiled query plans that exploit two JavaScript features: typed arrays and asm.js. Afterburner has the ability to support interactive data exploration via automatically-generated materialized views.

Data transformation R2D, RDF-to-Database, [63] aims to enable re-usability of relational tools on RDF data. R2D aims to transform RDF data, at run-time, into an equivalent normalized relational schema, thereby bridging the gap between RDF and RDBMS concepts and making the abundance of existing relational visualization tools available to RDF Stores. Linked Data Vizualization Wizard (LDVizWiz) [30] is a semi-automatic way for the production of possible visualization of linked data sets of high-level categories grouping objects that are worth viewing. It also associates the high-level categories with some very well-known vocabularies to facilitate the grouping of the information.

6 Experiments

In order to collect the raw data needed to create the knowledge database that is used by the decision model we have conducted a series of experimental analysis. The experiments were conducted using a laptop with an Intel(R) Core(TM) i7-4500U CPU at 1.80GHz, 4GB RAM memory connected to a 12Mbps home network connection.

Datasets. In order to compile a knowledge database able to support our decision model we need to include information from a wide variety of SPARQL endpoints. We have examined over 140 SPARQL endpoints. The majority of the endpoints were not available during Autumn of 2019 when the testing was conducted or were available sporadically allowing us to obtain answers to only a few endpoints. We have managed to collect some measurements from 55 endpoints. Ten of them had strict query time limits and/or were very slow in providing responses so only few characteristics were collected. Another ten of the endpoints were not supporting queries using the *strlength* function, so we collected the rest characteristics. Forty five endpoints were available throughout the testing period and were responding to queries in a consistent and timely manner. For these endpoints the majority of the characteristics were collected.

Metrics. We collected the data needed for the knowledge database which are the result size limit, the total number of unique predicates, the total number of unique subjects and objects, the most/least connected subjects and objects, the size of the dataset, the most/least used predicates, the min/max string length for subject,objects and predicates. Based on the above information we have calculated and added to the knowledge database the average node degree.

Methodology. We have carefully chosen specific SPARQL queries that allow us to collect the needed information, for some queries we have created different versions in order to overcome the restrictions of some endpoints regarding the use of specific keywords, such as *LIMIT* and *COUNT*. We have created a Python script that accesses the endpoints, provided as a list of URLs, and runs the queries, recording the results in files as text. The script records a list with failed queries and tries to re-run them at a later time in case it is due to temporally unavailability of the endpoint. The results are then inspected and provided that there are no issues with the responses they are inserted into the relational knowledge database. The queries used for the collection of the information are presented here:

- Result size limit. *SELECT ?subject ?predicate ?object WHERE {?subject ?predicate ?object}*
- Number of unique predicates. *SELECT (count(distinct ?predicate) as ?count) WHERE {?subject ?predicate ?object}*
- Number of unique subjects. *SELECT (count(distinct ?subject) as ?count) WHERE {?subject ?predicate ?object}*
- Number of unique objects. *SELECT (count(distinct ?object) as ?count) WHERE {?subject ?predicate ?object}*
- Number of predicates. *SELECT (count(?predicate) as ?count) WHERE {?subject ?predicate ?object}*
- Minimum appearances of predicates. *SELECT ?predicate (count(?predicate) as ?count) WHERE {?subject ?predicate ?object} GROUP BY ?predicate ORDER BY ASC(?count) LIMIT 1*
- Minimum appearances of objects. *SELECT ?object (count(?object) as ?count) WHERE {?subject ?predicate ?object} GROUP BY ?object ORDER BY ASC(?count) LIMIT 1*
- Maximum appearances of predicates. *SELECT ?predicate (count(?predicate) as ?count) WHERE {?subject ?predicate ?object} GROUP BY ?predicate ORDER BY DESC(?count) LIMIT 1*
- Maximum appearances of objects. *SELECT ?object (count(?object) as ?count) WHERE {?subject ?predicate ?object} GROUP BY ?object ORDER BY DESC(?count) LIMIT 1*
- Minimum string length for predicates. *SELECT ?predicate (strlen(str(?predicate)) as ?min) WHERE {?subject ?predicate ?object } ORDER BY ASC(?min) LIMIT 1*
- Minimum string length for subjects. *SELECT ?subject (strlen(str(?subject)) as ?min) WHERE {?subject ?predicate ?object } ORDER BY ASC(?min) LIMIT 1*
- Minimum string length for objects. *SELECT ?object (strlen(str(?object)) as ?min) WHERE {?subject ?predicate ?object } ORDER BY ASC(?min) LIMIT 1*
- Maximum string length for predicates. *SELECT ?predicate (strlen(str(?predicate)) as ?max) WHERE {?subject ?predicate ?object } ORDER BY desc(?max) LIMIT 1*
- Maximum string length for subjects. *SELECT ?subject (strlen(str(?subject)) as ?max) WHERE {?subject ?predicate ?object } ORDER BY desc(?max)*
- Maximum string length for objects. *SELECT ?object (strlen(str(?object)) as ?max) WHERE {?subject ?predicate ?object } ORDER BY desc(?max)*

Table 1. Experimental analysis of SPARQL endpoints

	Result size limit	Unique predicates	Unique subjects	Unique objects	Total predicates	Min subject degree	Min object degree	Max subject degree	Max object degree	Min predicate appearances	Max predicate appearances	Min strlen of predicate	Min strlen of subject	Min strlen of object	Max strlen of predicate	Max strlen of subject	Max strlen of object	Node Degree
Dbpedia [6]	10000	60649	2.4E+07	2.7E+07	4.4E+08	1	1	5	12856179	1	1.1E+08	47	28	32	47	404	51	0.12
URIBurner.com [27]	10000	75426	3.4E+07	3.8E+07	4.7E+08	1	1	426810	5696371	1	2.1E+07	35	2	1	56	76	576	0.15
OpenLink Virtuoso [20]	10000	4251	267167	347618	1533646	1	1	19748	90014	1	254487	3	1	1	121	331	23342	0.40
Allie Abbreviation Database [1]	10000	187	3.6E+07	3.7E+07	2E+08	1	1	184606	7642353	1	1E+08	29	12	1	76	49	4664	0.37
El Viajero's tourism [8]	10000	181	1019390	1127135	4631527	1	1	826	385491	1	1017691	29	12	1	75	358	65538	0.46
Lista de Encabezamientos de Materia [16]	30000	14	249328	537244	1757437	1	1	565	175758	29	351516	32	24	1	47	61	1651	0.45
OpenMobileNetwork [19]	40000	76	909837	1850866	2.3E+07	1	1	4847	357283	1	1.5E+07	30	30	1	73	200	671	0.12
Wiki Pathways [49]	100000	226	1139544	2012612	1.6E+07	1	1	4336	1230992	1	2028845	29	12	1	75	101	11239	0.19
BBC John Peel from DBTune [3]	None	25	76.255	122.136	349720	4	1	4	1	115	152301	47	30	1	47	71	2570	0.57
Magnatune from DBTune [17]	None	24	43.301	88.141	260487	16	15	16	15	265	43280	29	29	1	48	60	6468	0.50
STW Thesaurus for Economics [24]	None	7	48	99	143	2	1	48	45	1	48	30	2	2	62	55	58	1.03
Web-based Systems Group [11]	None	45	238	964	1662	1	1	31	79	2	233	30	72	1	63	158	6046	0.72
Alpine Ski Racers of Austria [2]	None	117	1267	1950	13441	1	1	145	1392	1	2745	29	32	32	74	218	1276	0.24
Datos [5]	None	32	677605	1936753	1.2E+07	1	1	18	322046	5	3000	36	49	63	55	268	282	0.22
Linked Open Vocabularies [15]	None	1269	178136	343748	861612	1	1	691	39154	1	157336	10	32	32	90	166	10538	0.61
Bio2RDF [4]	None	1	1.5E+07	144	1.4E+09	1	1	5750	125615	1	1.9E+08	39	29	1	66	65	93732	0.01
Open Data Thesaurus [18]	None	80	308	1351	3453	1	1	56	303	1	412	29	18	18	65	89	2617	0.48
OxPoints [21]	None	336	126315	313368	915052	1	1	21880	22978	1	126757	23	2	2	77	184	203920	0.48
Social Semantic Web Thesaurus [23]	None	159	12564	17844	127899	1	1	253	8879	1	14826	29	71	72	68	95	4937	0.24
Vacancies [28]	None	336	126315	313368	915052	1	1	21880	22978	1	126757	23	2	2	77	184	203920	0.48
Jamendo [13]	None	26	335.951	440.686	1385598	2	1	2	1	485	626242	29	29	1	60	94	33662	0.56
Geological Survey of Austria [10]	None	76	628	2936	7311	1	1	42	527	1	566	29	32	32	65	101	570	0.49
Isidore [12]	None	269	1.7E+07	70956381	4E+08	1	1	693397	26518	1	1.6E+08	29	12	1735	47	86	16492	0.22
DrugBank [7]	None	196	316950	1759602	3672531	4	1	568		2	251571	35	30	1	61	54	361	0.57
Revyu [22]	None	19	11105	21791	38359	1	1			1650	13005	47	35	1	63	81	790	0.86
UniProt [26]	None	214	1.2E+10	1.2E+10	5.5E+10	1	1			142047	9.5E+07	8	37	39	33	80	52	0.43
EventMedia [9]	10000	173	1.1E+07	5447856	1.1E+08	1	1	119	698903	1	2.9E+07			1	73		6	0.16
Camera dei deputati [14]	10000	357			2.4E+08	1	1		2204637	1	4.1E+07	37	44	4	43	69	46886	

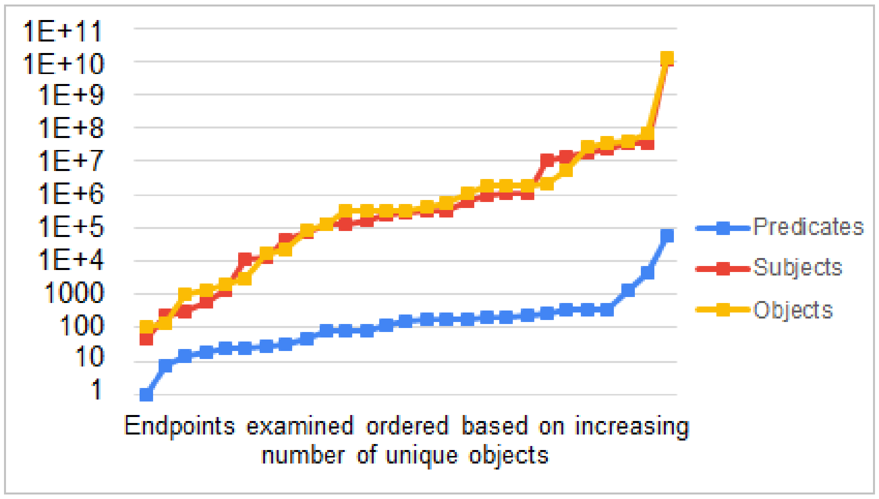


Fig. 8. The number of unique subjects, predicates and objects for the examined endpoints

Results. We present in Table 1 the results of the experimental analysis for some of the endpoints that we have evaluated. Aiming to keep the table readable and coherent we have added only the endpoints that have provided answers to all but one or two queries. Based on the results obtained we present here an in-depth analysis of the characteristics.

- Result size limit. From the 45 endpoints examined only 15 of them had any limit at the result size. The most popular limit was 10.000 but the value had a great deviation, from 500 to 100.000 elements and an average value of 18.700. This is a very important find as it sets the requirements regarding the volume of information that the user interface should be able to handle.
- Number of unique predicates. Endpoints have from 1 to 75.426 unique predicates, with an average value of 4.388. This deviation is very interesting with regard to the semantic differentiation of the datasets. Only a few datasets are focused on few semantic relationships between their entities, while most of them offer a higher variance.
- Number of unique subjects. The overall size of the dataset affects the number of unique subjects. The examined endpoints have from 48 to 11.520.028.275 unique subjects with an average value of 343.734.281.
- Number of unique objects. The overall size of the dataset affects also the number of unique objects. The examined endpoints have from 99 to 12.153.725.295 unique subjects with an average value of 295.390.891. While the minimum and maximum values are higher than the ones for the subjects, the average number of objects per endpoint is significantly lower from the average number of subjects. This is mostly due to the fact that few datasets contain descriptive information and free text while most of them depend on categories from the Semantic Web to describe the contained information.

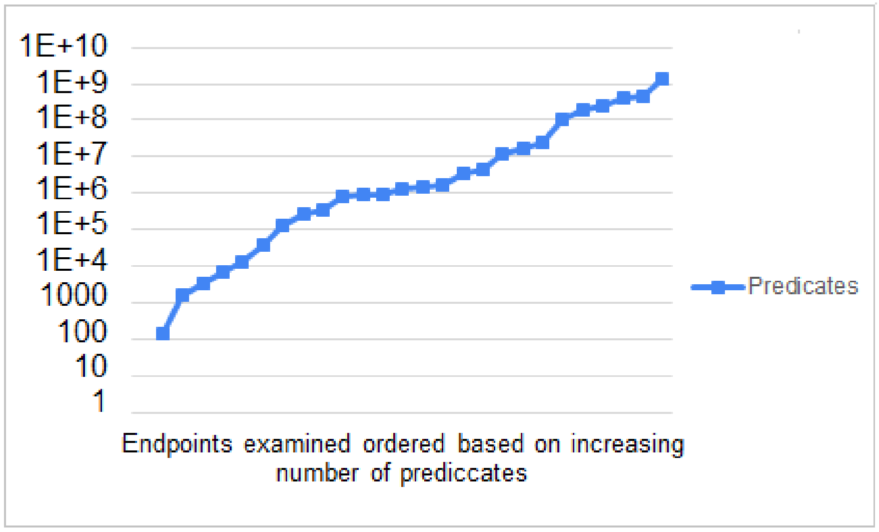


Fig. 9. The number of predicates for the examined endpoints

In Figure 8 we present the chart with all the values of unique subjects, predicates and objects for the examined endpoints.

- Number of predicates. The total number of predicates in a dataset represents the number of all the triplets of information and as a result the size of the dataset. Endpoints are dedicated to dataset ranging from 143 to 55.343.596.553 triplets, and an average value of 1.532.244.807. The difference between the minimum and maximum values is indicative of the diversity of the datasets available through endpoints. The results are presented in Figure 9.
- Minimum & Maximum appearances of predicates. The minimum appearances of a predicate range from 1 to 142.047 depending on the type of dataset while the maximum times a predicate is repeated in a dataset can be as high as 186.915.393 for an endpoint. These value show that endpoints provide access to diverse datasets, some are focused on specific relationships between entities, thus having few predicates that repeat many times, while others cover a wide range of topics and concepts, limiting the re-usability of terms.
- Minimum & Maximum appearances of subjects. The minimum appearances of a subject range from 1 to 16 while the maximum can be as high as 693.397 for an endpoint. As expected, here too the diversity of the datasets is affecting the deviation of the values.
- Minimum & Maximum appearances of objects. For the objects this deviation is even more pronounced. Here, the minimum appearances of an object range from 1 to 15 while the maximum can be as high as 12.856.179 for an endpoint. Similarly with the predicates, datasets that are providing information for specific scientific fields repeat key concepts and semantic terms multiple times.

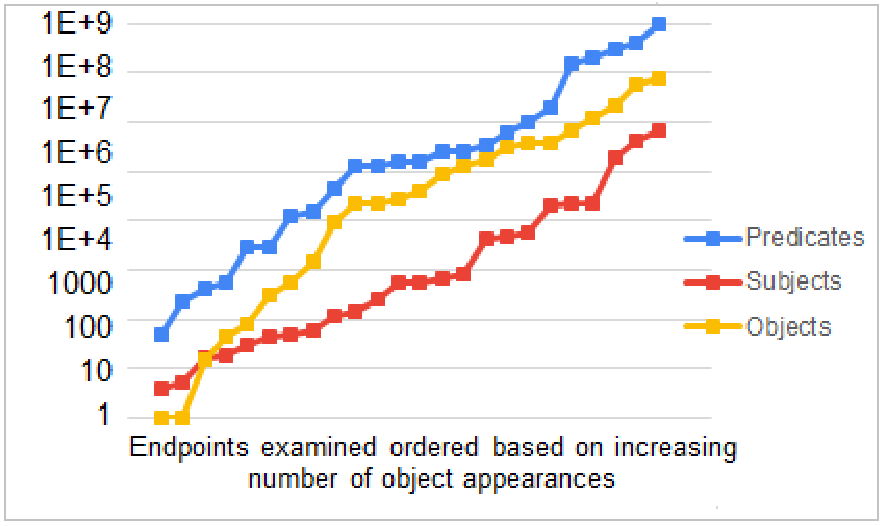


Fig. 10. The number of appearances for subjects, predicates and objects for the examined endpoints

In Figure 10 we present the chart with all the values of appearances for subjects, predicates and objects for the examined endpoints.

– Minimum & Maximum string length for predicates. The string length for the predicates ranges from 47 to 371 characters, with an average value of 62. These values are within the expected range as predicates are mostly URLs from the Semantic Web.
– Minimum & Maximum string length for subjects. The string length for the subjects ranges from 72 to 404 characters, with an average value of 87. These values are higher than the ones for the predicates as subjects sporadically include free text in addition to URLs from the Semantic Web.
– Minimum & Maximum string length for objects. The string length for the objects ranges from 72 to 203.920 characters, with an average value of 36.262. Given that objects are mostly descriptive and include a lot of free text these values are again expected.

In Figure 11 we present the chart with the maximum string lengths for subjects, predicates and objects for the examined endpoints and in Figure 12 the minimum string lengths.

– Average node degree. The average node degree for the datasets is 0.32, indicating that the datasets are not very connected and probably include independent sub-graphs. The distribution of the values is shown in Figure 13.

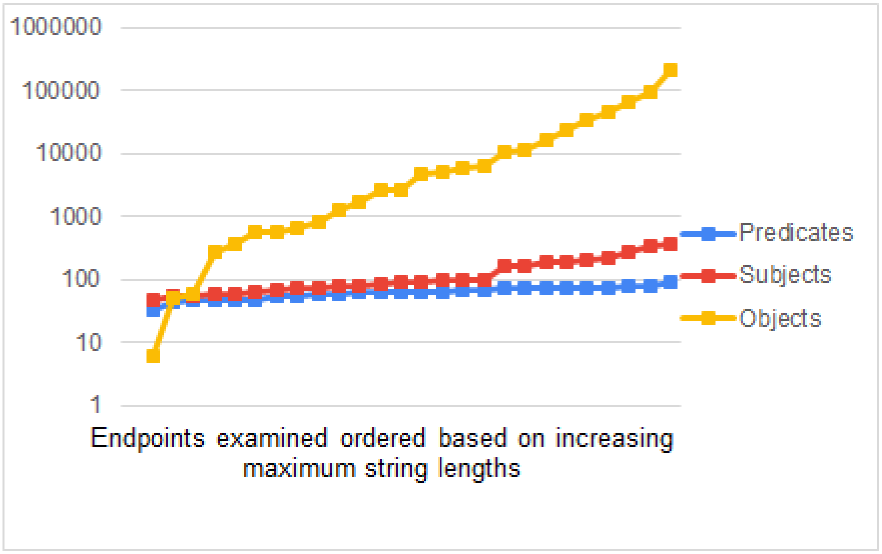


Fig. 11. The maximum string lengths for subjects, predicates and objects for the examined endpoints

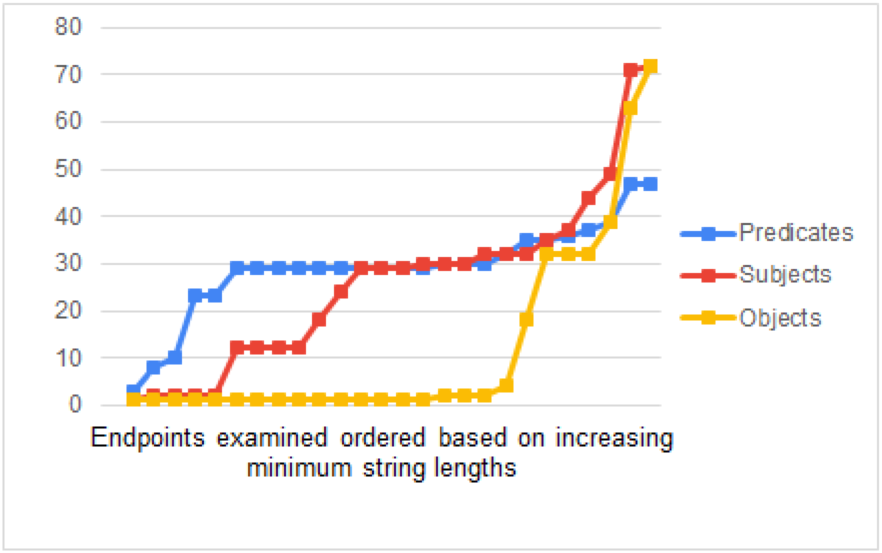


Fig. 12. The minimum string lengths for subjects, predicates and objects for the examined endpoints

Conclusions

In this paper, we present a novel system architecture that supports both expert and novice users with querying SPARQL endpoints, exploring and visualizing the query results in a dynamic way. Our system has been designed in a schema agnostic and data structure robust way that allows the visualization of diverse query results in a user-friendly and interactive way.

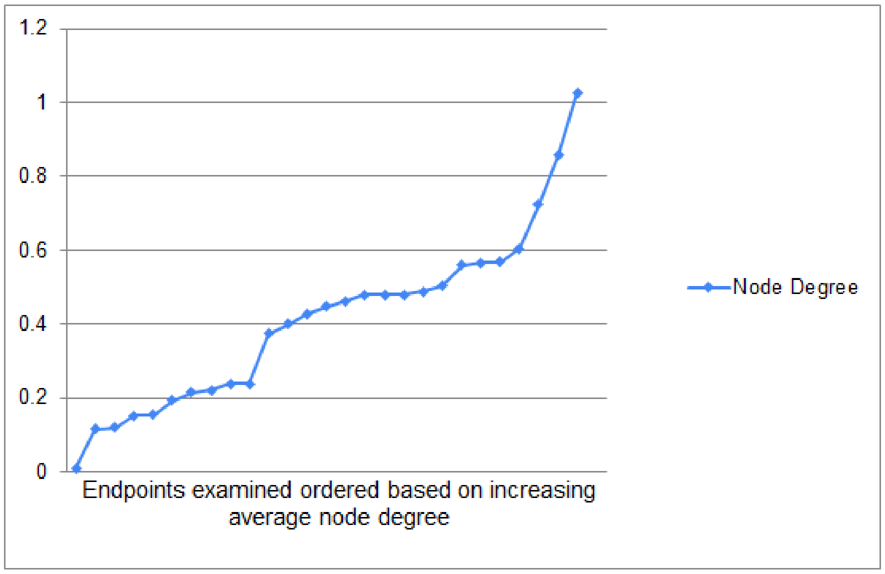


Fig. 13. The node degree for the examined endpoints

Acknowledgment

The experimental analysis presented in Section 6 was supported by Mr. Tara Venkata Sai Rahul Muktineni as part of his Master Project for the University of Ottawa.

References

1. Allie abbreviation and long form database in life. `http://data.allie.dbcls.jp/sparql` (2019)
2. Alpine ski racers of austria. `http://vocabulary.semantic-web.at/PoolParty/sparql/AustrianSkiTeam` (2019)
3. Bbc john peel sessions from dbtune. `http://dbtune.org/bbc/peel/cliopatria/yasgui/index.html` (2019)
4. Bio2rdf. `https://bio2rdf.org/sparql` (2019)
5. Datos. `http://datos.bcn.cl/sparql` (2019)
6. Dbpedia. `http://dbpedia.org/sparql/` (2019)
7. Drugbank. `http://wifo5-03.informatik.uni-mannheim.de/drugbank/snorql/` (2019)
8. El viajero's tourism dataset. `http://webenemasuno.linkeddata.es/sparql` (2019)
9. Eventmedia. `http://eventmedia.eurecom.fr/sparql` (2019)
10. Geological survey of austria (gba) - thesaurus. `https://resource.geolba.ac.at/PoolParty/sparql/lithology` (2019)
11. Information about the web-based systems group. `http://wifo5-03.informatik.uni-mannheim.de/dws-group/snorql/` (2019)
12. Isidore. `https://isidore.science/sparql` (2019)
13. Jamendo. `http://dbtune.org/jamendo/cliopatria/yasgui/index.html` (2019)
14. Linked open data camera dei deputati. `http://dati.camera.it/sparql` (2019)

15. Linked open vocabularies (lov). https://lov.linkeddata.es/dataset/lov/sparql (2019)
16. Lista de encabezamientos de materia as linked open. http://id.sgcb.mcu.es/sparql (2019)
17. Magnatune from dbtune. http://dbtune.org/magnatune/cliopatria/yasgui/index.html (2019)
18. Open data thesaurus. http://vocabulary.semantic-web.at/PoolParty/sparql/OpenData (2019)
19. Open mobile network. http://www.openmobilenetwork.org:8890/sparql (2019)
20. Openlink virtuoso. http://demo.openlinksw.com/sparql/ (2019)
21. Oxpoints (university of oxford). https://data.ox.ac.uk/sparql/ (2019)
22. Revyu. http://revyu.com/sparql/queryform (2019)
23. Social semantic web thesaurus. http://vocabulary.semantic-web.at/PoolParty/sparql/semweb (2019)
24. Stw thesaurus for economics. http://zbw.eu/beta/sparql-lab/ (2019)
25. Time ontology in owl. https://www.w3.org/TR/owl-time/ (2019)
26. Uniprot. https://sparql.uniprot.org/sparql (2019)
27. Uriburner.com. http://uriburner.com/sparql/ (2019)
28. Vacancies (university of oxford). https://data.ox.ac.uk/sparql/ (2019)
29. Ambrus, O., Möller, K., Handschuh, S., et al.: Konduit vqb: a visual query builder for sparql on the social semantic desktop. In: Workshop on visual interfaces to the social and semantic web (2010)
30. Atemezing, G.A., Troncy, R.: Towards a linked-data based visualization wizard. In: COLD (2014)
31. Battista, A.D.L., Villanueva-Rosales, N., Palenychka, M., Dumontier, M.: Smart: A web-based, ontology-driven, semantic web query answering application. Semantic Web Challenge **295** (2007)
32. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and analyzing linked data on the semantic web. In: ISWUIW. Citeseer (2006)
33. Berners-Lee, T., Fischetti, M.: Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor. DIANE Publishing Company (2001)
34. Bielefeldt, A., Gonsior, J., Krötzsch, M.: Practical linked data access via sparql: The case of wikidata. In: LDOW@ WWW (2018)
35. Bonduel, M., Rasmussen, M.H., Pauwels, P., Vergauwen, M., Klein, R.: Sparql-visualizer: A communication tool for collaborative ontology engineering processes
36. Brunetti, J.M., Auer, S., García, R.: The linked data visualization model. In: ISWC (2012)
37. Brunetti, J.M., Auer, S., García, R., Klímek, J., Nečaskỳ, M.: Formal linked data visualization model. In: Proceedings of ICIIWAS. ACM (2013)
38. De Araujo, S.F., Schwabe, D.: Explorator: a tool for exploring rdf data through direct manipulation. In: DOW2009 (2009)
39. Deligiannidis, L., Kochut, K.J., Sheth, A.P.: Rdf data exploration and visualization. In: Proceedings of the ACM CyberInfrastructure. ACM (2007)
40. El Gebaly, K., Lin, J.: In-browser interactive sql analytics with afterburner. In: ACM ICMD. ACM (2017)
41. Evergreen, S.D.: Effective data visualization: The right chart for the right data. Sage Publications (2019)
42. Graves, A.: Creation of visualizations based on linked data. In: International Conference on WIMS. ACM (2013)

43. Haag, F., Lohmann, S., Bold, S., Ertl, T.: Visual sparql querying based on extended filter/flow graphs. In: International Working Conference on Advanced Visual Interfaces. ACM (2014)
44. Haag, F., Lohmann, S., Ertl, T.: Sparqlfilterflow: Sparql query composition for everyone. In: ESWC. Springer (2014)
45. Haag, F., Lohmann, S., Siek, S., Ertl, T.: Visual querying of linked data with queryvowl. SumPre-HSWI@ ESWC (2015)
46. Heim, P., Ertl, T., Ziegler, J.: Facet graphs: Complex semantic querying made easy. In: ESWC. Springer (2010)
47. Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., Stegemann, T.: Relfinder: Revealing relationships in rdf knowledge bases. In: Semantic and Digital Media Technologies. Springer (2009)
48. Heim, P., Ziegler, J., Lohmann, S.: gfacet: A browser for the web of data. In: IMC-SSW'08. Citeseer (2008)
49. Helmich, J., Klímek, J., Nečaskỳ, M.: Visualizing rdf data cubes using the linked data visualization model. In: ESWC. Springer (2014)
50. Hoefler, P., Granitzer, M., Veas, E.E., Seifert, C.: Linked data query wizard: A novel interface for accessing sparql endpoints. In: LDOW (2014)
51. Huynh, D., Karger, D.R., Quan, D., et al.: Haystack: A platform for creating, organizing and visualizing information using rdf. In: Semantic Web Workshop. vol. 52 (2002)
52. Katayama, T.: D3sparql: Javascript library for visualization of sparql results. In: SWAT4LS. Citeseer (2014)
53. Kelleher, C., Wagener, T.: Ten guidelines for effective data visualization in scientific publications. Environmental Modelling & Software **26** (2011)
54. Kempf, D.A.O.: Stw-info (2019), `http://www.zbw.eu/en/stw-info/`
55. Klímek, J., Helmich, J., Nečaskỳ, M.: Payola: Collaborative linked data analysis and visualization framework. In: ESWC. Springer (2013)
56. Krommyda, M., Kantere, V.: Understanding sparql endpoints through targeted exploration and visualization. In: Graph Computing 2019 (2019)
57. Krum, R.: Cool infographics: Effective communication with data visualization and design. John Wiley & Sons (2013)
58. Lopez, V., Fernández, M., Motta, E., Stieler, N.: Poweraqua: Supporting users in querying and exploring the semantic web. Semantic Web (2012)
59. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In: ISWC 18. Springer (2018)
60. Martin, M., Abicht, K., Stadler, C., Ngonga Ngomo, A.C., Soru, T., Auer, S.: Cubeviz: Exploration and visualization of statistical linked data. In: World Wide Web. ACM (2015)
61. Mustafa, J., Dikaiakos, M.D.: Mashql: A query-by-diagram topping sparql towards semantic data mashups. University of Cyprus mjarrar
62. Nikolaou, C., Dogani, K., Bereta, K., Garbis, G., Karpathiotakis, M., Kyzirakos, K., Koubarakis, M.: Sextant: Visualizing time-evolving linked geospatial data. Journal of Web Semantics (2015)
63. Ramanujam, S., Gupta, A., Khan, L., Seida, S., Thuraisingham, B.: R2d: A bridge between the semantic web and relational visualization tools. In: IEEE ICSC. IEEE (2009)
64. Rietveld, L., Hoekstra, R., et al.: Man vs. machine: Differences in sparql queries. In: ESWC (2014)

65. Russell, A., Smart, P.: Nitelight: A graphical editor for sparql queries (2008)
66. Russell, A., Smart, P.R., Braines, D., Shadbolt, N.R.: Nitelight: A graphical tool for semantic query construction (2008)
67. Skjæveland, M.G.: Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In: ESWC. Springer (2012)
68. Tufte, E.R.: The visual display of quantitative information, vol. 2. Graphics press Cheshire, CT (2001)
69. Wang, Y., Zhu, M., Qu, L., Spaniol, M., Weikum, G.: Timely yago: harvesting, querying, and visualizing temporal knowledge from wikipedia. In: International Conference on Extending Database Technology. ACM (2010)
70. Weise, M., Lohmann, S., Haag, F.: Extraction and visualization of tbox information from sparql endpoints. In: EKAW. Springer (2016)
71. Wirtschaft, L.I.: Zbw (2019), `http://www.zbw.eu/de/`
72. Yamaguchi, A., Kozaki, K., Lenz, K., Wu, H., Kobayashi, N.: An intelligent sparql query builder for exploration of various life-science databases. In: IESD@ ISWC (2014)
73. Zhu, Y.: Measuring effective data visualization. In: International Symposium on Visual Computing. Springer (2007)
74. Zviedris, M., Barzdins, G.: Viziquer: a tool to explore and query sparql endpoints. In: ESWC. Springer (2011)