MAKING LINKED-DATA ACCESSIBLE: A REVIEW

Omar Mussa
School of Computer Science and Informatics
Cardiff University, UK
mussao@cardiff.ac.uk

Omer Rana
School of Computer Science and Informatics
Cardiff University, UK
ranaof@cardiff.ac.uk

Benoît Goossens
School of Biosciences
Cardiff University, UK
goossensbr@cardiff.ac.uk

Pablo Orozco-terWengel
School of Biosciences
Cardiff University, UK
orozco-terwengelpa@cardiff.ac.uk

Charith Perera
School of Computer Science and Informatics
Cardiff University, UK
pererac@cardiff.ac.uk

August 17, 2023

ABSTRACT

Linked Data (LD) is a paradigm that utilises the Resource Description Framework (RDF) triplestore to describe numerous pieces of knowledge linked together. When an entity is retrieved in LD, the associated data becomes immediately accessible. SPARQL, the query language facilitating access to LD, contains a complex syntax that requires prior knowledge due to the complexity of the underlying concepts. End-users may experience a sense of intimidation when faced with using LD and adopting the technology into their respective domains. Therefore, to promote LD adoption among end-users, it is crucial to address these challenges by developing more accessible, efficient, and intuitive tools and techniques that cater to users with varying levels of expertise. Users can employ query formulation tools and interfaces to search and extract relevant information rather than manually constructing SPARQL queries. This paper investigates and reviews existing methods for searching and accessing LD using query-building tools, identifies alternatives to these tools, and highlights their applications. Based on the reviewed works, we establish 22 criteria for comparing query builders to identify the weaknesses and strengths of each tool. Subsequently, we identify common usage themes for current solutions employed in accessing and searching LD. Moreover, we explore current techniques utilised for validating these approaches, emphasising potential limitations. Finally, we identify gaps within the literature and highlight future research directions to further advance LD accessibility and usability for end-users.

***K*eywords** Linked-Data, Linked open data, Visual querying, SPARQL, Query builders, Visualization

1 Introduction

The emergence of Linked Data (LD) has sparked competition among various organisations seeking to disseminate their data in a widely adoptable, machine-readable format, typically through the utilisation of Resource Description Framework (RDF) triples as the primary means of distribution [33, 23]. The interconnected nature of these entities promotes accessibility and machine-readability, consequently leading to an accelerated growth of published data, which now encompasses billions of triples. However, this vast expansion has simultaneously heightened the complexity for humans attempting to navigate and extract useful information from such data sources [33, 23, 62].

Linked-Data is also referred to as Linked Open Data (LOD) when open data sources are employed. The availability of LOD through SPARQL endpoints has led to the receipt of millions of daily queries from both human and machine requesters [19, 106, 88]. Despite data machine-readability enabled by LD, humans still constitute a substantial portion of its consumers [88, 20]. For accessing LD, SPARQL is the W3C recommended Query Language [58]. It is a precise and powerful query language that enables users to specify exact relationships that match explicit patterns and constraints, even though LD contains billions of triples and relationships. However, learning SPARQL can be challenging initially, as it is complex and creates a high cognitive load [115, 62, 7]. Thus, mainstream users require direct assistance to access LD using SPARQL, although experts find it less daunting. Nonetheless, writing accurate SPARQL queries necessitates an understanding of the underlying ontology that defines the scheme of relationships and terms between distinct entities. This includes the namespaces, vocabularies, and data names. Unfortunately, numerous SPARQL endpoints lack the "unique name assumption" and have inadequate or nonexistent documentation [119, 27, 17]. Consequently, developers may have to guess the appropriate ontology and rely on chance to stumble upon the correct names. Therefore, even SPARQL experts may face difficulties in creating SPARQL queries to extract data from an unfamiliar SPARQL endpoint [62].

Given the complexity of SPARQL, what is the rationale behind its usage? The complexity of SPARQL can primarily be attributed to the intricate structure of the underlying data and the complexity of the queries intended. The language is highly expressive, but it is not the only option for querying knowledge graphs, with languages like Cypher or Gremlin being employed for the same purpose. However, these languages are also not suitable for Lay-users [5, 98, 58]. SPARQL is the leading standard for querying RDF triple stores and is endorsed by the World Wide Web Consortium (W3C). Additionally, it excels in querying semantically linked, heterogeneous, and distributed datasets [5, 58].

The development of a user-friendly interface to assist users in creating SPARQL queries and effectively exploring data is urgently needed. There have been numerous efforts to develop a SPARQL Query Builder (QB) Interface. Several efforts have been made to create a SPARQL Query Builder (QB) Interface, which is an End-User Development (EUD) tool that enables users to construct SPARQL queries using graphical interface components [9, 14]. EUD tools are crafted to fulfil the end-user requirements and skills and to provide an alternative that overcomes the difficulty of using SPARQL [101]. However, many of these interfaces still have steep learning curves and require a profound understanding of the concepts behind LD and the ways in which data is connected [113, 78]. While some tools aim to enhance experiences within the Semantic Web community, others are tailored to support experts in domain-dependent settings. In contrast, many mainstream or lay users have expressed a preference for tools to query LD that do not necessitate previous experience with SPARQL or Semantic Web.

This paper thus explores techniques to allow access to LD for users with no SPARQL experience as a means of improving user experience. The main focus is on using the user interface to generate SPARQL queries or to navigate LD more efficiently. Thus, the ability to form a SELECT SPARQL query was essential for a tool to be chosen. While some of the tools examined may not be able to generate other types of queries, such as CONSTRUCT and ASK, this is not seen as a limitation for this study. The paper offers four main contributions:

- Investigating different research approaches utilised for searching Linked Data.
- Examining the current solutions being employed for accessing and searching Linked Data, with identification of common usage themes.
- Exploring current techniques used for validating these approaches, with emphasis on highlighting potential limitations.
- Analysing gaps in the existing literature and identifying potential future research directions.

The rest of this paper is organised as follows: Section 2 provides a motivating example that illustrates how wildlife researchers would benefit from having a query builder to extract information from LD. Section 3 explains the search methodology used. Section 4 discusses the categorisation scheme used with our evaluation framework, while Sections 5 and 6 explore tools and approaches available that offer access to LD, as well as introduce SPARQL Query Builders.

Section 7 will then discuss current semantic web solutions and their usage themes. Section 8 explores the used evaluation approaches with some research validation limitations. Section 9 then analyses and discusses some of the findings, allowing Section 10 to focus on the emergent research challenges and limitations. Section 11 then concludes the work by offering some final thoughts.

2 Motivating Example

As a motivation for our work, let us assume the scenario of a team of wildlife and bioscience researchers engaged in data collection within forest environments. Conventionally, the collected data is recorded in a CSV format. Subsequent to its collection, the data is typically archived in its raw format, compressed as a zip file, and then uploaded to a shared repository for potential future usage or to be disseminated among interested researchers.

However, the characteristics of the data are inherently heterogeneous, primarily owing to the variability in the research focus. Consequently, there is a significant lack of systematic means to render this archived data readily accessible and searchable for other researchers when it is maintained in its conventional format. This presents a considerable challenge to the effective and efficient use of such data in further research.

Suppose these researchers decide to utilise LD by disseminating their data in a more accessible format, which can accommodate the heterogeneous structure of their data, similar to the methodologies adopted in Forest Observatory[1] initiative. As illustrated in Figure 1, the conceptualisation of the Forest Observatory proposes that LD can be composed of a wide variety of data types. These data may originate from various projects and data sources. These diverse data types are semantically integrated, thereby forming a large knowledge graph that fulfils the requirements of bioscience research. However, would it necessitate a preliminary understanding of LD to access this data effectively? Furthermore, what would be the most efficient approach for information extraction?

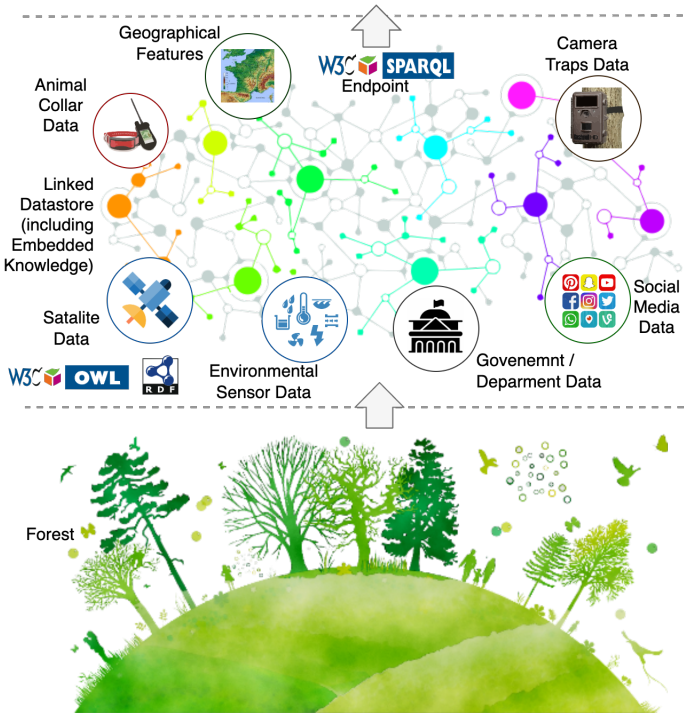

Figure 1: An illustration of the conceptual design for Forest Observatory.

The inherent inconsistency of their data introduces an additional layer of complexity. Even if given the correct SPARQL query as a template suited to a specific dataset, the utility of this template cannot be guaranteed for a different dataset. The following two paragraphs will demonstrate how SPARQL query builders can significantly aid in LD information retrieval without necessitating any prior knowledge about SPARQL or other semantic web technologies.

[1] `https://www.forest-observatory.org/`

Consider a scenario where researchers attempt to identify specific animals and samples drawn from a particular area among tons of datasets. Crafting such a query would present a significant challenge, even for experts. However, by employing a user-friendly interface, individuals can simply mark a region on a map and initiate a search without any understanding of the underlying technologies.

Similarly, connecting various attributes, perhaps located in entirely distinct archived datasets or even related to a different project – such as correlating water levels with the presence or behaviour of certain species – could prove exceedingly difficult without the aid of a specialised interface. This complex task can be made significantly more manageable by enabling the user to select these datasets and coordinate the results based on time and location. Even if the time format varies across datasets, an intelligent interface can effortlessly identify this inconsistency and explain it to the user. Upon successful retrieval, the data should be ready for further processing and analysis, fulfilling the researchers' intentions for its use.

The above methodology can find application across various fields, thereby promoting the adoption of semantic web technologies across multiple domains. Users lacking technical expertise stand to gain significantly from such technology, negating the need for them to comprehend complicated query languages and concepts such as SPARQL, IRIs and LD.

3 Methodology

As the scope of this review paper can be relatively enormous, we have determined several stages and strategies to identify the relevant work to shape the purpose of this paper. The initial step involves using Google Scholar to locate all potentially relevant work and determine the primary keywords and data sources. Then, we conducted a manual search on these sources. In addition, we have performed citation mining on all of the papers by doing backwards and forward citation searches (snowballing). Figure 2 summarises the search stages as well as the selection technique.

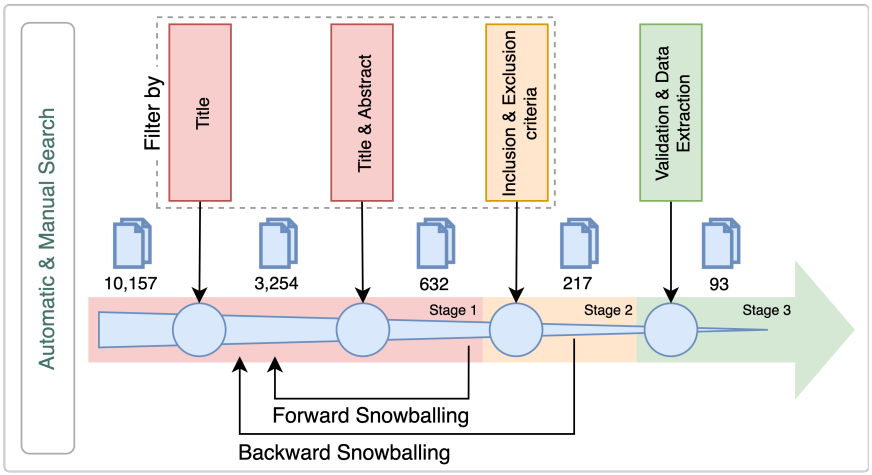

Figure 2: Demonstration of the search stages with the selection strategy.

3.1 Search Strategy

Influenced by [75] proposed strategy, the search strategy was divided into three stages. The first step was dedicated to locating all potentially relevant papers. Initially, we were concerned only with the title to filter out irrelevant papers. Then, we used the paper's abstract to create the initial list of potentially relevant papers. We started by identifying primary keywords that include "SPARQL", "Semantic Web", and "Linked-Data" to be combined with words like "Query Formulation", "Visual", "Query Builder", and "Natural Language". We have used Google Scholar to perform an automatic search on these keywords without focusing on any specific publisher.

Typically, the results of using single keywords were overwhelmingly large, mainly out of the research scope. Combining the keywords using AND operator has helped lower the results into more relevant papers. In addition to identifying potential papers, we have identified relevant sources to be used for manual search. Google Scholar was also used to "forward snowballing" to find more candidate papers and identify more sources.

Then, we have manually searched the identified sources using the keywords combination. The identified sources were specialised journals or conference proceedings such as "Semantic Web journal (by IOS Press)", "International Working

Conference on Advanced Visual Interfaces", and "International Conference on Semantic Systems". For section 7, we have reviewed 33 out of 145 paper that was published as in-use or resource during 2022 - 2015 in the International Semantic Web Conference.

3.2 Selection Approach

We have identified relevant papers using only the title and abstract in the initial stage. Therefore, we had to read the entire paper in the second stage to filter out the irrelevant papers. Thus, we have defined inclusion and exclusion criteria that each paper must satisfy to stay on the list. First, we must check against the exclusion criteria to filter out all matching cases. Then, if it passes, we will check if it satisfies at least one of the inclusion criteria to stay on the list.

3.2.1 Inclusion and Exclusion Criteria

The selection criteria were defined to ensure that the final list will fall within this paper domain. The exclusion criteria were: (i) Papers not written in English. (ii) Requiring to type SPARQL manually. (iii) Not fully published paper (Only abstract or a poster). (iv) Does not discuss LD. These criteria must be avoided to be qualified to the inclusion criteria. The inclusion criteria were designed to ensure that the papers were of quality and relevant to this review. The inclusion criteria were: (i) Introduce UI for querying LD (ii) discuss approaches for LD visualisation. (iii) Introduces a tool or a solution to access LD. (iv) Facilitating the process of exploring LD. Satisfying one of the inclusion criteria means that the paper is qualified as a selected paper.

3.3 Validation and Data Extraction

The third stage mainly compares the selected paper using the manual and automatic search with the known papers. Also, this will include checking the references and performing backwards snowballing. Thus, we repeated the selection approach for any new papers and updated the list of selected papers. Then, to extract the data efficiently, we have created a list containing the paper ID, title, year, publication source, tool name (if available), categorisation, visualisation approach, type of validation (if applicable) and validation sample size.

4 Categorisation and Comparison

To enable a meaningful comparison of tools and to identify similarities and patterns of use, it was necessary to categorise the tools based on specific criteria and establish a framework for evaluating their usability. This framework was developed to enable clear differentiation between the tools and facilitate a comprehensive analysis.

4.1 Categorisation

Upon conducting an in-depth review of diverse techniques for accessing and searching linked data, we have identified various tools that can be categorised as Query builders, designed for constructing SPARQL queries, while others enable the access and consumption of data in distinct approaches. The categorisation of query builders was motivated by previous studies conducted by [49, 32, 78, 113, 76], with the aim of arranging them according to their querying approach. On the other hand, the categorisation of other user interfaces was suggested by multiple papers, which are mentioned individually in each relevant section.

Initially, we classified the query builders into three major groups: Form-based, Graph-based, and Natural Language-Based. Subsequently, we organised alternative user interfaces into the following groups: SPARQL Assistants, Linked-Data Browsers, Semantic Keyword-Based Search, Question Answering, and SPARQL to Natural Languages. Figure 3 illustrates the categorisations of the main identified approaches.

4.2 Query Builders Comparison Criteria

Our analysis of query formulation approaches and usage patterns, combined with insights from previous research by [76, 32, 33**?**], led us to identify 22 criteria that could serve as the basis for a query builder usability framework, enabling comparison among different approaches. Each criterion was selected to fulfil a specific purpose. Below is a list of all the criteria, along with a brief explanation for each.

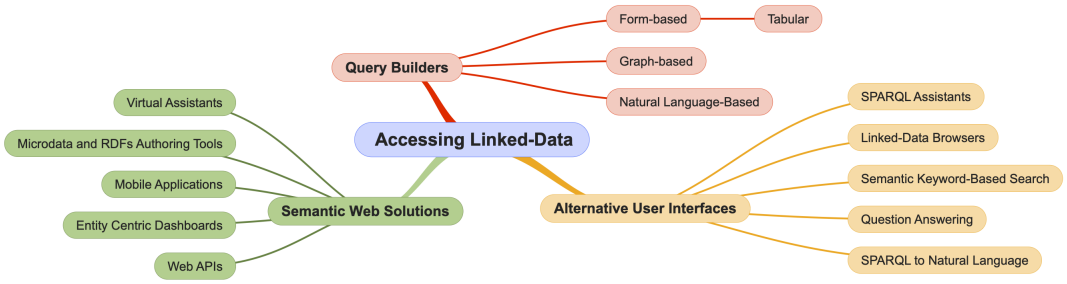

Figure 3: A diagram that illustrates the reviewed tools' classification for obtaining data from SPARQL endpoints in addition to the usage themes for the semantic web solutions.

Criterion 1 Target Users: Through a comprehensive review of the existing literature and approaches in the field, our analysis has identified three distinct user categories that could potentially derive benefits from the utilisation of linked data searching tools and data extraction approaches. The identification of these user categories provides valuable insights into the target audience for such tools, emphasising the importance of customising the tools to meet the specific requirements and preferences of each user group. The three user categories are as follows:

- **Lay-users**: These users lack any prior knowledge or understanding of SPARQL or Semantic Web.
- **Tech-users**: These users possess an understanding of the Web Semantic domain and LD operations. They might even be capable of reading SPARQL and comprehending its significance. However, they have not engaged directly in writing SPARQL queries.
- **Expert-users**: These users are well-versed in SPARQL and can create complex SPARQL queries. They have previously worked with RDF and other Semantic Web technologies.

Criterion 2 Environment: The term "environment" refers to the means by which the software can be accessed, which can take one of two forms: a web application that can be accessed via any browser, offering a high degree of portability, or a desktop application that typically provides greater robustness and stability but requires an installation and may have specific access requirements, such as a particular operating system or packages.

Criterion 3 Visual metaphor: The visual representation of the interface is a reflection of a particular underlying concept. Despite the classification of similar interfaces based on their approach to querying, they present various visualisation ideas. For instance, some Form-Based query builders resemble a conventional form, such as Konduit VQB, while others adopt alternative visualisations like SparqlFilterFlow's use of the FilterFlow Model. It is crucial to distinguish between these visualisations to assess their influence on the user.

Criterion 4 Text Search: This criterion pertains to the capacity of a query builder to utilise text-based input for searching entities or resources or constructing queries. While not all query-building tools prioritise natural language, a significant number of them offer the option of keyword-based searching for identifying resources from an endpoint. Some tools, such as natural language QBs, leverage sentence-based searching to formulate queries. Consequently, this criterion can be either a keyword search or a sentence search.

Criterion 5 Dataset: In evaluating a tool, one aspect taken into account is its ability to access an arbitrary endpoint or if it necessitates a specific dataset. The majority of query builders are generic and have been constructed to access any triplestore through the SPARQL endpoint. However, certain query builders have been tailored to access particular datasets, such as Wikidata or DBpedia.

Criterion 6 Reflect Result Size: In assessing the effectiveness of the filters used during query construction, it is crucial to consider how users perceive the resulting size. For example, one approach to addressing this issue is demonstrated in work by [55], which alters the line width between entities to indicate the decrease in result size after a filter is applied. Thus, this feature holds importance to the user and would prove advantageous if it were to be implemented in some way by the other tools we have identified.

Criterion 7 Is evaluated? In assessing a tool, it is crucial to consider whether it has been subjected to user evaluation to gauge its usability and effectiveness. In some cases, tools have undergone manual validation of results without actual user involvement, which we deem as not fulfilling the evaluation criteria.

Criterion 8 Programming Language: To ascertain the underlying technology of a tool, it is vital to determine the programming language utilised in its development. Such knowledge is critical for potential reusability or integration with other tools that employ similar technologies.

Criterion 9 Automated suggestions: This evaluation criterion focuses on the tool's capability to provide suggestions to the user as they type or select entities from a list in order to decrease complexity, enhance productivity, and, crucially, aid in exploring the dataset without requiring knowledge of the underlying ontology.

Criterion 10 Filtering: The majority of tools permit users to incorporate filters into their queries, with certain variations in terms of expressivity. Nonetheless, some tools lack this functionality and solely enable the selection or exposure of RDF triple relations without the ability to specify filters directly. This criterion only considers whether the tool allows for the addition of filters or not without taking into account the expressivity of these filters.

Criterion 11 Examples: The provision of an example or playground query to users is a crucial feature that promotes learning by example, facilitating a smoother learning curve and faster adoption of the tool. The criterion for this feature is met only if the tool offers users example queries for their use.

Criterion 12 History: Enabling users to track their query-building history and roll back any errors is a critical feature, especially when constructing complex queries, as the interface can become convoluted. This function empowers users to experiment and engage more deeply with the query without fear of making mistakes and having to reconstruct the query from scratch. As such, it is a valuable tool that promotes user confidence and encourages greater exploration of the tool's capabilities. This criterion is only met if the user has the ability to undo an action or review their history list and return to a specific point in their timeline of activity.

Criterion 13 Multiple perspectives: This criterion is focused on evaluating the tool's ability to offer users multiple perspectives of their queries. The interface should provide various views that allow the user to visualise the query in different formats. An instance of this feature in action could be a tool that allows the user to visualise their query using an RDF Graph format but also offers an alternative tabular format for representing the same query. Additionally, the tool should offer the option to show or hide additional views that represent the query or enable users to switch the interface to a different view as needed. This feature can help users gain a better understanding of their queries from different angles, which can lead to more effective query construction and data analysis.

Criterion 14 & 15 View and Edit SPARQL: The ability to expose the generated SPARQL query is a crucial aspect of any query formulation tool, particularly for those with advanced technical knowledge and expertise. Furthermore, the option to manually edit the query represents an additional level of functionality that can significantly enhance the usability of the tool for expert users, enabling them to expedite the query formulation process and manually make modifications as required. It should be noted that while this functionality is not present in some query formulation tools, it constitutes a fundamental point of differentiation between query formulation and conventional data exploration.

Criterion 16 Save Query: This criterion is only satisfied if the tool provides a means for saving the query, such as the ability to save it as a file or within the tool account or browser cookies. Moreover, the saved query must be easily accessible for the user to retrieve as needed. This is a valuable feature that allows users to create and test multiple queries and provides the option to reuse them in the future.

Criterion 17 Share Query: Enabling users to share their queries with others, whether through a file, account, URL, or any other means, is a valuable feature that enhances the user's ability to collaborate and explore results with others. This capability promotes greater engagement among users and provides a quality feature that can facilitate learning and more effective use of the tool. If a tool permits query saving but lacks the ability to share it with other users, then this criterion is not satisfied.

Criterion 18 Non-Empty Results: Preventing users from obtaining empty results enables them to concentrate on relevant data, thereby enabling a more satisfying and efficient experience. Implementing such a feature enhances the user's confidence in the tool and saves time and resources. For instance, when applied in conjunction with auto-complete functionality, this feature could hide a URI if its selection yields an empty result. This strategic approach reduces user

frustration and facilitates the data exploration process. This criterion is only satisfied when the tool is designed to prevent the occurrence of empty results.

Criterion 19 Export Results: Exporting results in conventional formats, such as CSV, is not commonly incorporated in many SPARQL query builders. This feature holds significant importance for users seeking to extract data from a sizeable dataset and utilise it for their own purposes at a later time. This criterion is satisfied if the user has the ability to export the results in any format other than the RDF JSON serialisation.

Criterion 20 Access Resource IRI: This evaluation criterion refers to the user's capacity to access additional information about a specific resource by clicking on its IRI (Internationalised Resource Identifier). This functionality is vital as it enables users to gain a better understanding of the data and perform more sophisticated queries.

Criterion 21 Supporting Inference: This assessment refers to the tool's ability to retrieve data that is connected to a specific resource through inferencing correlations based on entity relationships. For instance, if a resource describes 'New York City' and another resource describes 'The Big Apple', they can be linked using OWL:sameAs, as both refer to the same city. If the tool supports this feature and is used to extract data about 'New York City', it should detect the URI of 'The Big Apple' and present it to the user as well. This functionality is often associated more with natural language processing (NLP) tools than other types of tools.

Criterion 22 Results Visualisation: This evaluation criterion is concerned with the format in which results retrieved from SPARQL endpoints are presented to the user. Typically, these results are provided in the form of an RDF JSON serialisation, which is not easily readable by humans due to its large size and complexity. To facilitate user understanding and interpretation, the results should be presented in a human-readable format, such as a tabular format. This involves converting a resource URI into a more easily readable version. This criterion is considered fulfilled if the results can be displayed in any format other than the original JSON format.

5 Visual Query Builders

This section, explores the tools available to obtain data from SPARQL endpoints. As mentioned in section 4.1, the tools categorisation was based on the visual approach of the data retrieval process (see Figure 3). The Visual Query Builders classification was inspired by [49, 78, 113] to represent the visual interface based on its querying approach.

Query builders serve as sophisticated instruments designed for searching linked data, presented through visual interfaces that empower users to construct personalised queries. The primary objective of these tools is to enhance the efficiency of information retrieval by providing an effortless means of extracting the desired information, outperforming the performance of conventional methods. By providing a more user-friendly environment, query builders cater to diverse users, including those with limited technical expertise. Therefore, these tools not only improve the user experience but also promote the broader adoption of linked data and semantic web technologies, contributing to a more interconnected and accessible data ecosystem. Subsequently, we present an extensive array of tools, which are grouped into three main categories, each encompassing a diverse spectrum of functionalities and use cases.

5.1 Form-based

Utilising conventional form-based interfaces in the design of query-building tools is an effective approach, capitalising on user familiarity to reduce the learning curve. Most of the tools reviewed in this section employ common user interface elements, such as drop-down lists, text fields, and buttons, to create an intuitive and user-friendly search interface. This familiarity allows users, especially those who may not be well-versed in the process of querying LD, to more easily navigate the interface and construct queries that serve their intended purpose.

5.1.1 Konduit VQB

The Konduit VQB [4] interface presents an approach for formulating "CONSTRUCT" or "SELECT" queries rooted in either "schema" or "instance" concepts. This query builder was developed as a Java desktop application, accounting for its traditional form-based interface appearance. The design caters to expert and tech users, streamlining the process of creating SPARQL queries. The "Schema-based" interface capitalises on the hierarchical nature of SPARQL, simplifying the underlying components by employing a top-down organisation and field indentation to represent parent-child relationships. In contrast, the "Instance-based" interface utilises the RDF triple structure (Subject, Predicate, and Object), facilitating more precise queries with less abstraction, though potentially too cryptic for lay users. Despite

lacking user evaluation, the tool incorporates several user-friendly features, such as keyword searches and automated suggestions within drop-down menus.

5.1.2 BioGateway Query Builder App

The BioGateway App [67] is a desktop application purposed for querying the BioGateway SPARQL endpoint. This application shares similarities with the Konduit VQB Schema-based interface, such as employing multiple drop-down lists within a form-based interface. Nevertheless, the BioGateway App does not adopt the parent-child structure observed in Konduit VQB; rather, each line embodies an RDF triple, which can be connected to subsequent lines using predefined variables (e.g., Set A, Set B). A notable distinction between the BioGateway App and Konduit VQB lies in the former's provision of predefined examples, fostering learning through example. Despite this advantage, the application is more domain-specific and necessitates a degree of familiarity with the BioGateway ontology and SPARQL, potentially posing challenges for even technical users.

5.1.3 Wikidata Query Service (WQS)

The WQS [2] serves as Wikidata's official query instrument for executing SPARQL queries. It incorporates a query builder, the "Query Helper," which supports users in creating or refining SPARQL queries when their proficiency in SPARQL is limited [88] (see Figure 4). The Query Helper utilises a form-based interface, predominantly featuring drop-down lists, and derives its conceptual framework from "Filters" and "Show" design. This interface assists users in pinpointing items that align with specified filters, whilst permitting the addition of optional matches through the "Show" section to be displayed upon data retrieval. The auto-generated SPARQL query is presented on the right-hand side of the interface, affording tech and expert users the opportunity to manually modify the query and witness the consequent alterations within the Query Helper. In contrast, lay users may perceive this feature as unwarranted and potentially overwhelming [78].

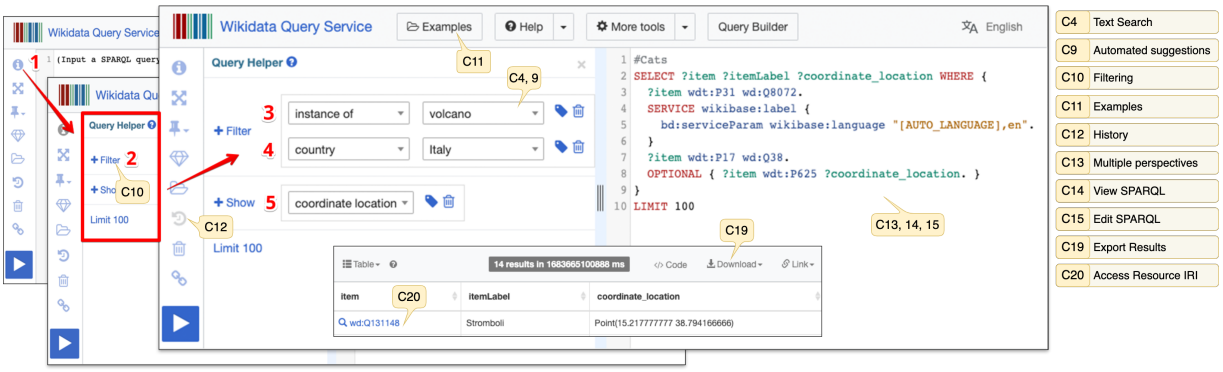

Figure 4: An example of using the Query Helper interface in Wikidata Query Service to generate SPARQL to extract all of the "Italian Volcanoes with its coordinates if exist". (1) Clicking on the button to show Query Helper. (2) Clicking on Filter. (3) Writing "Volcano" as the wanted entity and select "instance of" as the subject. (4) Then, adding Italy which by default will select country as its subject. (5) Finally, clicking on "Show" to add "coordinate location" as an optional property. The numerical value associated with the callout corresponds to the usability criteria that the highlighted element satisfies.

5.1.4 VizQuery

VizQuery [3], a straightforward query builder, operates by matching rules corresponding to RDF triple patterns. Its interface aids users in identifying items that adhere to the chosen rules. For instance, to locate all "Cats" within a dataset, a user must select the property "instance of" and the item "house cat" (see Figure 5). In contrast to WQS, this tool does not accommodate the addition of optional patterns or the construction of more complex structures. However, as with WQS, VizQuery incorporates auto-complete functionality and offers a predefined example, thereby augmenting the user experience.

[2] https://query.wikidata.org/
[3] https://hay.toolforge.org/vizquery/

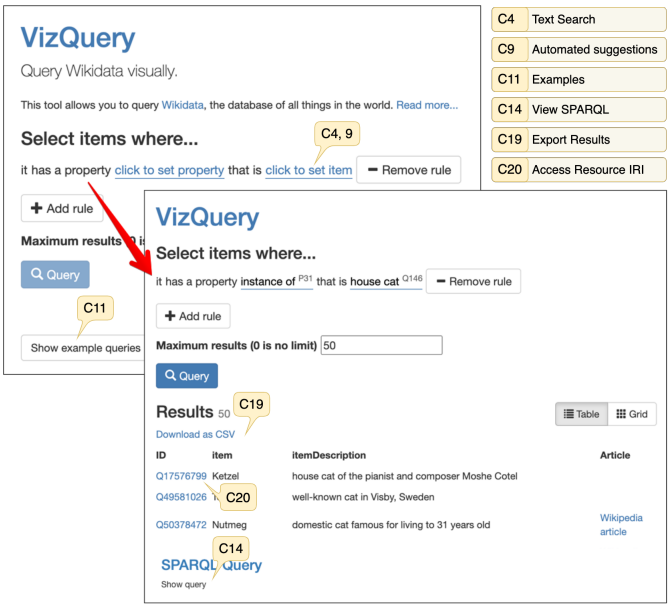

Figure 5: An example of using the VizQuery interface to find all indoor Cats in Wikidata dataset by selecting the property "instance of" and the item "house cat". The numerical value associated with the callout corresponds to the usability criteria that the highlighted element satisfies.

5.1.5 SparqlFilterFlow

One of the highly interactive query builders is SparqlFilterFlow [55], which is based on the concept of FILTER/FLOW MODEL that was introduced in 1993 as an SQL query builder. The interface combines a form-based design with a graphical representation, fostering a more intuitive understanding of relationships between entities. Users initiate by defining an entity and subsequently applying filters to refine the results [53]. Coloured links interconnect all interface components, facilitating a top-down data flow, while the varying thickness of links between graph entities represents the quantity of data present. A user study conducted by the author revealed that participants successfully completed all tasks and were capable of comprehending and rectifying their errors to achieve accurate output. The path thickness feature played a crucial role in helping participants adjust their input, as it conveyed the volume of data being retrieved and informed their responses. Although the original prototype presented by [54] was a desktop application, the authors have developed a web-based demo [4] implementing the same approach, albeit with limited features (see Figure 6).

5.1.6 Simplod

Simplod, as proposed by Jares and Klimek [70], is a web application designed to facilitate the extraction of data in bulk from LOD in a CSV format. This tool necessitates the incorporation of the LOD schema in RDF Turtle format to formulate the query. The Dataset schema view, which is depicted on the left portion of the user interface, presents the entities within a UML Class diagram to illustrate the relationships. Conversely, the schema list view, situated on the right segment of the interface, showcases the schema in a list form, with checkboxes indicating the inclusion of elements in the search.

It is important to note that Simplod is limited to generating SELECT SPARQL queries and lacks customisation or filtering features. Consequently, the primary utility of Simplod lies in the exploration of LOD and the bulk retrieval of results. In their evaluation, they reported a System Usability Scale (SUS) score of 63.125 for users lacking prior RDF experience, which falls below the average score of 68. However, users with RDF expertise demonstrated a higher score of 72.5, implying that Simplod necessitates training and is thus most suitable for tech users.

5.1.7 Tabular Query Builders

Capitalising on users' familiarity with spreadsheet applications, Tabular Query Builders often display linked data in a tabular format, enabling users to apply filters and constraints to refine the results similarly to conventional spreadsheet

[4] `http://sparql.visualdataweb.org/sparqlfilterflow.php`

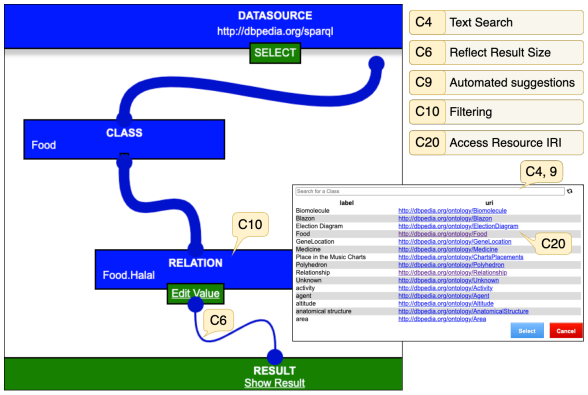

Figure 6: An example of using SparqlFilterFlow's online demo to get all of the entities that is Food and also Halal. The thickness of the lines reflect the results size. The numerical value associated with the callout corresponds to the usability criteria that the highlighted element satisfies.

Table 1: The usability criteria for the reviewed Form-based SPARQL Query Builders. "E": Expert-users, "T": Tech-users, "L": Lay-users, "All": All users, "Dt": Desktop-App, "Web": Web-based App.

Usability Criterion	Falcons Explorer [29]	VizQuery	SparqlFilterFlow [55]	LDQW [62]	SPARQLViz [24]	PepeSearch [60]	BioGateway App [67]	Konduit VQB [4]	ExConQuer [7]	WQS [78]	Simplod [70]
1. Target Users	E,T	All	All	All	E,T	T,L	E	E,T	E,T	E,T	T
2. Environment	Web	Web	Dt/Web	Web	Dt	Web	Dt	Dt	Web	Web	Web
3. Visual metaphor	Facet Tabular	Forms	FilterFlow	Tabular	Wizard	Forms	Forms	Schema Instance	Facet Tabular	Forms	Schemas
4. Keyword search	✓	✓	✓	✓		✓	✓	✓	✓	✓	
5. Dataset	Any	Wikidata	DBpedia	Any	Any	Any	BioGateway	Any	Any	Wikidata	Any
6. Reflect Size			✓								
7. Is evaluated?			✓	✓		✓				✓	✓
8. Prog. Language	JS	JS	C#/JS	JS	Java	JS	Java	Java	JS	JS	JS
9. Auto. suggestions	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
10. Filtering	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
11. Examples		✓					✓			✓	
12. History										✓	
13. Multiple persp.	✓			✓	✓	✓	✓	✓	✓	✓	✓
14. View SPARQL		✓			✓		✓	✓	✓	✓	✓
15. Edit SPARQL								✓	✓	✓	✓
16. Save query					✓		✓	✓			✓
17. Share query					✓		✓				✓
18. Non-Empty R.				✓							
19. Export Results		✓			✓				✓	✓	✓
20. Resource IRI	✓	✓	✓			✓			✓	✓	
21. Inference											
22. Visualisation	Table	Table Grid	List	Table Mind Map Chart	Table IsaViz	Table	Table Graph Net.	Table	Table	Table Chart Graph	Yasgui

software. These builders are categorised under form-based query builders due to their incorporation of traditional form elements, such as drop-down menus and text fields, albeit in a more tabular structure. Notable examples of Tabular Query Builders encompass Linked Data Query Wizard (LDQW) [62], ExConQuer [7] and Falcons Explorer [29].

5.1.7.1 Linked Data Query Wizard (LDQW)

LDQW [62] is a SPARQL query builder that leverages user familiarity with search engines and spreadsheet applications to reduce the learning curve and enhance user abstraction from the LD concept. The LDQW prompts users to initiate with a keyword, similar to a conventional search engine, and subsequently directs them to another page displaying preliminary results in a tabular layout. Users can drill down into the data by applying filters based on the available

properties and incorporating new columns into the table to represent some of the concealed data. Furthermore, users can visualise the data in a mind map format or using predefined charts. LDQW also provides users with the opportunity to examine the generated SPARQL queries accompanied by runtime statistics, which is a remarkable feature for expert and tech users. Overall, the usability study outcomes were promising, albeit revealing several design shortcomings that warrant improvement.

5.1.7.2 ExConQuer

Employing a tabular layout, ExConQuer [7] emerges as another SPARQL query builder that skillfully enhances user familiarity with its interface. Contrasting with LDQW's approach of presenting a table containing initial results, ExConQuer exhibits the chosen subject's properties, enabling users to select from these properties to apply data filters and refine the results. Consequently, ExConQuer adopts the Faceted-Browsing concept to prevent overwhelming users, dividing the interface into three primary stages. The first stage entails selecting the data source, followed by the user selecting a single subject, referred to as a concept. Ultimately, users obtain all data pertaining to that subject, with the option to choose additional properties for matching during the data query.

Prior to executing the query, a text box containing the auto-generated SPARQL is displayed, allowing users to modify the SPARQL before manually executing it. According to the authors, this feature aims to assist users in learning about and acclimating to SPARQL and RDF while enhancing the experts' use experience. The tool was assessed by expert and tech users in comparison to manually creating SPARQL queries. These users were tasked with performing identical tasks with and without the tool, subsequently rating their experiences in both instances. The findings revealed that the tools proved advantageous for all users, particularly those with limited SPARQL experience.

5.1.7.3 Falcons Explorer

Falcons Explorer [29], an LD Browser created on top of "Falcons Object Search," a search engine that is designed to search for entities on LD. Assigning this tool to this section might seem odd, given that it resembles a conventional LD browser with support for faceted browsing. However, it integrates a built-in query builder, which shares some commonalities with LDQW. Users initiate their search by entering an entity name into the search box. The resulting data can be displayed in a tabular layout, enabling users to select a specific entity, which subsequently directs them to the corresponding entity page containing all related properties and information.

The results table parallels a spreadsheet in its functionality, offering multiple filtering techniques and column management options. Besides the tabular representation, the tool also supports data presentation in a relational format, similar to relational database design. This relational arrangement permits users to employ typical relational operations such as Projection and Joins. As the interface offers various underlying options and customisable filters, it may require some familiarity with LD and, therefore, may appear confusing to lay users.

5.1.8 Summary

By leveraging the user's familiarity with conventional forms, form-based QBs are generally designed to be accessible and suitable for a broader range of users. Konduit VQB, BioGateway App, WQS, VizQuery, and SparqlFilterFlow all share a common attribute in that they depend on the visualisation of the RDF triple structure through the employment of pre-populated drop-down lists. Consequently, users are required to select an item and specify filters for the final output. Although both Konduit VQB and BioGateway App demand a certain level of experience in SPARQL, WQS, VizQuery, and SparqlFilterFlow offer more simplified and intuitive designs. VizQuery can be further characterised as a simplified implementation of the WQS design, as VizQuery rules adhere to a similar filter schema (refer to Table 1).

Besides the familiarity with conventional forms, LDQW, ExConQuer, and Falcons Explorer have further enhanced the user experience in querying and exploring linked data by relying on facet search and users' familiarity with spreadsheets. After retrieving the preliminary results, users are given the ability to apply a variety of filters, thereby facilitating the adjustment and customisation of the query to align with their specific preferences and requirements.

5.2 Graph-based

From the initiation of LD, data have been visualised in some graphical formats. Students have also been taught to think of an RDF triple as a node with an arrow pointing to another node. The first node represents the "Subject", the second node will present the "Object", while the arrow itself is the "Predicate" (see Figure 7). Therefore, many of the published

SPARQL Query Builders attempted to follow this principle as a way to simplify the idea of the LD it describes. For the purposes of this paper, these types of query builders are described as Graph-Based Query Builders.

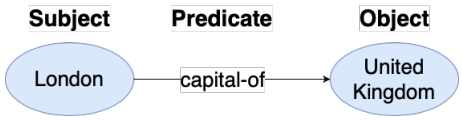

Figure 7: An example of RDF basic triple pattern.

5.2.1 Graphical Query Language (GQL)

One of the earliest attempts to create SPARQL Query Builder is the GQL Tool [17]. The authors have followed "Ontology Definition Metamodel" [5] to present the query in the form of UML class diagrams. The tool is a desktop application that allows the user to manually select a class from a list of the available classes generated from the LD ontology; these classes are then linked together. The user can filter the results by adding predefined properties, such as setting a class property to be less than a certain numeric value or to match a specific string. The authors thus claim that the tool can be used to create very complex queries.

5.2.2 SPARQLinG

Rather than representing RDF in UML class diagrams as seen in the GQL Tool, [66] have introduced a unique graphical representation called RDF-GL. The authors have explained the symbolic notation used in terms of how it simplifies the SPARQL query visualisation. They also introduced a tool called SPARQLinG, a Java-based desktop application that creates SPARQL queries using RDF-GL. The tool provides a wide area to users that they can use to drag and drop any RDF-GL components they wish to construct the required graph that will restore the required data. The current tool only converts RDF-GL to SPARQL, and does not operate the other way around. SPARQLinG may thus be easier for Experts, and Tech-users than GQL Tool, as the graphical representation in RDF-GL is more intuitive than that in UML.

5.2.3 SPARQLING

SPARQLING [16] is another tool that aims to produce SPARQL queries intuitively, and it should be noted that while the name is similar to SPARQLinG, they are different tools. SPARQLING is a platform to construct SPARQL queries by drawing them over the interface. What distinguishes SPARQLING from similar tools is that it uses GRAPHOL ontologies [31]. GRAPHOL is a visual representation of the OWL 2 ontologies that represent the data as an ER-Diagram [80]. The tool is a web-based application with three main sections. The left-hand side, which is the most significant part, is dedicated to the GRAPHOL view, while the right-hand side is divided into the SPARQL view and SPARQL visualisation view. SPARQLING was not evaluated in this work and was customised to query the 'Ontology-based Data Management'[81] systems.

5.2.4 iSPARQL

One of the most well-known query builders is the OpenLink interactive SPARQL Query Builder (iSPARQL) [6]. iSPARQL is a powerful tool that can autogenerate SPARQL from graphical representations, though it also supports the manual creation of SPARQL. iSPARQL relies on the RDF triple graph, which it uses to generate SPARQL. The tool can also be used to browse results within the interactive interface in tabular format. In addition, users can use it to query any standard SPARQL endpoint. Lay-users must understand the LD fundamentals, such as variables must start with '?', and a specific entity must begin with its prefix like 'sioct:Weblog'. Furthermore, Experts and Tech-users must familiarise themselves with the required SPARQL endpoint [109].

5.2.5 NITELIGHT

NITELIGHT[109, 102, 103] is a stand-alone client-side Web application built entirely using JavaScript. As with iSPARQL, NITELIGHT is a SPARQL query builder that relies on the RDF basic triple pattern, making each graph entity either a Subject or an Object connected using a Predicate. INITELIGHT and iSPARQL have many similarities, including allowing the user to perform complex SPARQL queries. However, in comparison to iSPARQL, NITELIGHT has additional novel features, such as allowing the user to browse the ontology in a tree-like structure, as well as

[5] `https://www.omg.org/odm/`

[6] `https://virtuoso-catalogue.d4science.org/isparql/`

offering a more realistic and attractive graphical representation [109]. The text-based SPARQL query produced is tightly coupled to the graphical view, which means that any change in the graph will immediately be reflected in the textual format. However, NITELIGHT still expects the user to be familiar with SPARQL and the endpoint's ontology.

5.2.6 ViziQuer

ViziQuer [119**?**] is a Graph Query Builder that was created to facilitate exploring unfamiliar SPARQL endpoints. The user can drag entity boxes around and draw paths between them. Then, the user can define each entity by choosing the class it represents from a drop-down list. The user can also click on a link between the classes and use "Connect Classes" to get the suggested predicate. For example, if the first node is 'Patient' and the second node is 'Physician', one of the suggested predicates will be 'familyDoctor', which forms a link between these two entities. Compared to the previously discussed Graph Query Builders, this is a straightforward interface that requires little prior knowledge about SPARQL. Each entity encapsulates all of its own data, and thus by right-clicking on each entity, users can select the appropriate options, such as displaying attributes and applying filters. Finally, users can choose to execute SPARQL from the graph directly or to generate the SPARQL and then manually review it. Figure 8 demonstrate using the tool to extract the names of all male patients with a family doctor. This figure also exposes a potential issue with the generated SPARQL with regard to a duplicate for the gender triple existing (one was declared as OPTIONAL). This may disclose some bugs when generating the queries from the diagrams. The tool is web-based and publicly available [7].

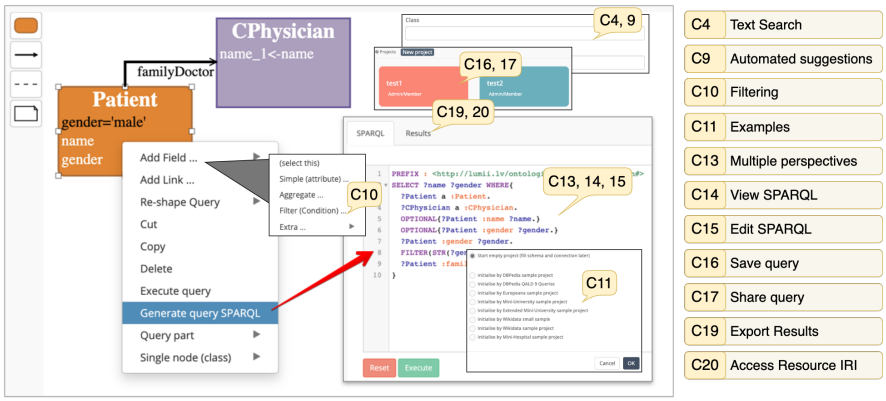

Figure 8: An example of using ViziQuer to generate SPARQL query that will extract the names of all male patients who have a family doctor. The numerical value associated with the callout corresponds to the usability criteria that the highlighted element satisfies.

5.2.7 QueryVOWL

QueryVOWL [56] is one of the simplest graph query builders with a visual representation that is based on an intuitive graphical notation known as Visual Notation for OWL Ontologies (VOWL)[56]. WebVOWL [84, 85] was an early attempt to visualise ontologies using VOWL, and QueryVOWL uses a similar interface to visualise the ontology while adding additional features to construct the query. The user can construct a query by typing and by selecting properties from the text box. Each of the selected properties is rendered as a single circle labelled with the property name. The user can draw a line between these circles to represent the matching relationship (the predicate). The potential predicates are then given as a list of options, and the user must select the most appropriate. On selecting each node, multiple options on the side menu appear, including options to add filters or change the node.

Each node has a numerical value that shows the possible objects (results) that match the current query. Typically, when adding more and more relationships and filters, the results will narrow, which means this number will decrease. The user can then click on "Show Query" to view the generated SPARQL query, which will exactly match the graphical representation. The tool thus provides a high level of abstraction for users, who can use it without any previous knowledge about SPARQL or the endpoint ontology. However, they must become familiar with VOWL and the tool itself. In addition, constructing SPARQL in QueryVOWL appears to be more time-consuming than manually creating a query or using other tools such as WQS [78].

[7] `https://viziquer.lumii.lv`

5.2.8 OptiqueVQS

OptiqueVQS [110] is a visual SPARQL query builder built for users with minimal IT skills. The interface is based on widgets, and it consists of three primary widgets: the list widget on the left-hand side, the diagram widget on the top and the form widget on the right-hand side. Users must begin by using the list widget to choose a concept as the starting node. A node will then be displayed in the diagram widget, and the form widget will be activated to manipulate the node details, such as adding filters and displaying the node variables. The authors divided the interface into three simple sections to be used in conjunction to simplify the query construction process. At the same time, they have focused on the industrial domain by generating two use cases for industrial users [110]. The usability study showed promising outcomes when this builder was used in the industrial field; however, constructing SPARQL is limited to a tree-like query pattern, omitting any other query patterns [110, 78].

5.2.9 RDF Explorer

RDF Explorer [113] is a query builder created for Lay-users with no expertise with SPARQL or LD. Its visualisations focus on the graphical representation of the query rather than the RDF triple pattern. The authors used WQS as the baseline system for comparison, and the demo thus generates a SPARQL query that must be used with Wikidata to be tested. RDF Explorer is a drag-and-drop interface where the user must begin by typing the resource name and drag the one that matches its query. Then, the user has to create a variable and link it with another node to perform a query pattern. There is no control over variable appearance in the query. The generated SPARQL is then placed within an editable text area, though any SPARQL manual alteration will not affect the tool, which may confuse the user.

The user study showed that RDF Explorer generally obtains better results eventually. While at first, users seem to struggle, the more complex the query, the better they perform in comparison to when using WQS[113], suggesting that any difficulty using the tool initially tends to decrease over time. All of the participants were undergraduate Computer Science students with no experience with SPARQL, though they were more likely to have advanced computer knowledge. They were generally slower in three of the five tasks. Furthermore, the RDF Explorer does not support exploring the results directly from the interface, which is unfortunate, as displaying results has been previously proven to be a suitable approach to assist users in constructing the correct queries [54].

5.2.10 Summary

Generally, most of the efforts in creating SPARQL Query Builders are focused on Graph-based visualisation, as this reflects the most natural behaviour when thinking about LD. The majority of these visualisations are based on the RDF triple pattern, due to the actual structure of the data, as in iSPARQL, NITELIGHT, ViziQuer, RDF Explorer, and OptiqueVQS. Other tools have proposed alternative visualisations in order to increase the abstraction and simplify the design. For example, the GQL tool uses UML, while SPARQLinG has created a custom visualisation called RDF-GL. While the use of the RDF triple may seem natural for Expert-users, greater abstraction seems to encourage Lay-users engagement. Table 2 summarises the reviewed graph-based query builders. Table 2 summarises the reviewed graph-based query builders.

5.3 Natural Language-Based

When it comes to Lay-users, the utilisation of natural language is considered to be the most convenient method of communication. Therefore, multiple tools have been developed to facilitate the generation of SPARQL queries through natural language input. The majority of these attempts have concentrated on developing question-answering systems that yield either YES/NO or single-statement responses. Nevertheless, this section exclusively discusses those tools that are capable of generating SPARQL queries and enabling users to retrieve multiple results through a graphical interface.

5.3.1 Querix

Querix [72] is an NL-Based SPARQL Query Builder that resolves ambiguities by asking users for intent clarification. Querix requires users to start their questions with an interrogative word or a verb to help construct the query skeleton, which is then used to classify RDF triple pattern in the sentence. To improve the query results, users should also specify the domain county and ontology. If there are too many possible queries with high scores emerge, a list with all potential user intents is displayed, and the user is asked to choose one. Thus, Querix achieves a high accuracy rate in comparison to other NL tools [71].

Table 2: The usability criteria for the reviewed Graph-based SPARQL Query Builders. “E”: Expert-users, “T”: Tech-users, “L”: Lay-users, “All”: All users, “Desktop”: Desktop-App, “Web”: Web-based App, “N/A”: signifies the absence of support for connecting to any SPARQL endpoint or dataset.

Usability Criterion	NITELIGHT [109]	GQL tool [17]	LuposDate [52?]	SPARQLinG [66]	SPARQLING [16]	iSPARQL	QueryVOWL [56]	RDF Explorer [113]	ViziQuer [?]	OptiqueVQS [110]
1. Target Users	E,T	E,T	E,T	E,T	E	E	All	All	E,T	All
2. Environment	Web	Desktop	Web	Desktop	Web	Web	Web	Web	Web	Web
3. Visual metaphor	RDF-Triple	UML	RDF-Triple	RDF-GL	GRAPHOL	RDF-Triple	VOWL	RDF-Triple	UML	RDF-Triple
4. Keyword Search							✓	✓	✓	
5. Dataset	Any	OpenLink Virtuoso	Any	N/A	N/A	Any	Any	Wikidata	Any	Any
6. Reflect Result Size							✓			
7. Is evaluated?				✓	✓		✓	✓		✓
8. Programming Language	JS	C#	Java	Java	JS	JS	JS	JS	JS	Java
9. Automated suggestions			✓				✓	✓	✓	✓
10. Filtering	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
11. Examples					✓			✓	✓	
12. History					✓					✓
13. Multiple perspectives		✓			✓	✓	✓		✓	✓
14. View SPARQL	✓		✓	✓	✓	✓	✓	✓	✓	✓
15. Edit SPARQL					✓	✓			✓	✓
16. Save query				✓		✓			✓	✓
17. Share query				✓		✓			✓	✓
18. Non-Empty Results								✓		
19. Export Results		✓							✓	✓
20. Access Resource IRI						✓		✓	✓	✓
21. Supporting Inference										
22. Results Visualisation		Table	Table			Table	Table		Table	Table Pivot Timeline

5.3.2 NLP-Reduce

NLP-Reduce [73] is an NL-Based tool for users with no experience with SPARQL. The tool allows users to type in full questions to be converted to SPARQL. The question will be cleaned and reduced into tokens that are then used over three distinct rounds to identify matching triples. Ultimately, the highest-scoring triple will be selected and will be joined with the other identified triples to generate the SPARQL query. Thus, users can also use non-complete sentences or even keywords [73]. NLP-Reduce has been proven to act as a rapid query-building tool, particularly in comparison to certain similar tools [71]. However, there are some limitations to the type of questions NLP-Reduce can answer. Users can not ask questions that require comparisons or ask questions with groupings such as “Who is the best football player in each team?” [2]. Therefore, the tool may not allow users to generate queries that fully express their needs.

5.3.3 QUICK

While most of the keywords NL-based query builders lack the expressivity to capture the user intention, QUICK [117] is an NL-based tool that specifically aims to provide users with a more satisfying query construction experience. The user types keywords in the search area; then, the tool displays all of the possible queries on the right-hand side. The queries are represented as text but with a graphical illustration to help aid understanding. Users must then choose the most appropriate query. If none of the options on the right matches the user intention, the tool provides a list with “Construction Options” on the left as a means of changing the query perspective. Once the user chooses any of these options, the possible queries on the right will change to match the user selection. QUICK thus incrementally supports the user to construct more accurate queries. However, this can cause the query options list to become lengthy and tedious to work through, while Lay-users may become overwhelmed by the graphical notation. QUICK also cannot construct ’cyclic graph’ queries, being limited to ’acyclic graph’ query patterns [117].

5.3.4 SPARKLIS

SPARKLIS [45] is an NL-Based Query Builder that converts complete sentences into SPARQL queries. The user can clearly understand the generated SPARQL query by looking into the constructed sentence. SPARKLIS initially offers the user three boxes: Entities, Concepts and Modifiers. The user must select options from these boxes to start building the sentence. The listed options then change to match the user selections, making forming a sentence in SPARKLIS a

guided process based on selection and related suggestions. Therefore, the user cannot create uncompleted sentences, which is beneficial for Lay-users. However, due to these restrictions, user freedom is decreased [2]. Accordingly, the user will require more time to create queries than when using other tools [78, 2]. The tool also showed some loading difficulties due to the frequent updates in the suggestions boxes [78].

5.3.5 Summary

Natural language query builders have been empirically proven to be easier to use. Nevertheless, the inherent variability of language presents significant challenges in accurately identifying users' intent and providing relevant answers to their queries. NLP-Reduce addresses this issue by accepting either keywords or complete questions and attempting to identify corresponding triples, irrespective of the sentence structure, ultimately selecting the triple with the highest degree of similarity.

In contrast, Querix seeks to overcome this challenge by prompting users to choose from a list of questions that closely align with their initial query. Similarly, QUICK employs an interactive approach by offering users an array of graph-based options, seeking their assistance in formulating the correct query. SPARKLIS, however, adopts a distinct approach by constraining the sentences users can generate through the implementation of drop-down lists. This controlled environment effectively mitigates the issue of language variability, thereby enhancing the query builder's accuracy and reliability. Table 3 summarises the reviewed NL-based query builders.

Table 3: The usability criteria for the reviewed Natural Language-based SPARQL Query Builders. "E": Expert-users, "T": Tech-users, "L": Lay-users, "All": All users, "Desktop": Desktop-App, "Web": Web-based App.

Usability Criterion	Querix [72]	NLP-Reduce [73]	QUICK [117]	SPARKLIS [45]
1. Target Users	T, L	T, L	All	All
2. Environment	Desktop	Desktop	Web	Web
3. Visual metaphor	Forms	Forms	RDF Triple	Forms
4. Text Search	Sentence	Keyword	Keyword	Sentence
5. Dataset	Any	Any	Any	Any
6. Reflect Result Size				
7. Is evaluated?	✓	✓	✓	✓
8. Programming Language	Java	Java	Java	JS
9. Automated suggestions				✓
10. Filtering	✓	✓	✓	✓
11. Examples				✓
12. History				✓
13. Multiple perspectives				✓
14. View SPARQL	✓	✓	✓	✓
15. Edit SPARQL				✓
16. Save query				
17. Share query				
18. Non-Empty Results				✓
19. Export Results				✓
20. Access Resource IRI				✓
21. Supporting Inference				
22. Results Visualisation	Table	Table	Table	Table Map Slideshow

6 Alternative User Interfaces

This section investigates a variety of tools designed to enhance access to LD, diverging from conventional SPARQL query builders by shifting the focus away from direct query construction. The emphasis is placed on data browsing and identifying precise answers. The discussion encompasses several categories of tools, including SPARQL Assistants, LD Browsers, Keyword-Based Search Systems, and Question Answering Systems. The section concludes with an examination of SPARQL to Natural Language tools, providing a comprehensive overview of these distinct approaches to facilitating LD access.

6.1 SPARQL Assistants

Assuming the user already has experience with SPARQL, query editing interfaces offer increased flexibility and enable more precise querying. These tools aid experts in familiarising themselves with the SPARQL endpoint by providing suggestions and autocomplete features. Notable examples of such tools include SPARQL Assist [89], YASGUI [100], and WQS [88].

SPARQL Assist is a web-based SPARQL query editor designed for less experienced SPARQL endpoint users. This tool supports users by presenting all potential options corresponding to the cursor position within a query. For instance, if a user types 'Ca' as a 'Subject', the interface displays a list of subjects beginning with 'Ca'. The list also includes descriptions for each item, where applicable. Consequently, constructing a manual SPARQL query evolves into a guided process, obviating the need for prior knowledge of the endpoint ontology. Moreover, the tool supports multi-language terms by examining resource labels to identify possible matches.

Both YASGUI and WQS exhibit similarities to SPARQL Assist, albeit with fewer constraints and guiding features, rendering them more appropriate for experienced users. YASGUI functions as a general-purpose SPARQL editor, designed for utilisation as an endpoint editor, whereas WQS serves exclusively as an editor for Wikidata. Neither tool supports multi-language terms, unlike SPARQL Assist. However, as previously discussed, WQS offers a form-based SPARQL query builder that is closely integrated with the text editor, mirroring the majority of user input. In contrast, YASGUI delivers an extensive array of methods for exploring results and can be incorporated with any query builder for result retrieval and visualisation, as shown in Simplod [70].

6.2 Linked-Data Browsers

The massive number of relationships a single resource may have in LD makes the data visualisation and navigation appear to be complicated. Thus, many efforts have been made to encounter this challenge through the development of LD Browsers. These browsers enable users to navigate and explore LD in a manner akin to web browsing, offering an accessible and intuitive experience. Notable examples of LD Browsers include Tabulator [21], Humboldt [77], Ozone Browser [26], Parallax [68], gFacet [61], and Rhizomer [25].

While LD Browsers have made significant strides in enhancing data visualisation and navigation, they often lack the capability to form specific queries, which can be a critical drawback when dealing with knowledge graphs containing millions of triples. Consequently, answering complex questions or extracting precise information from such large datasets can be both tedious and time-consuming. Some LD Browsers, however, have attempted to address this limitation by integrating advanced querying features into their interfaces such as in Falcons Explorer (discussed in section 5.1.7.3).

6.3 Semantic Keyword-Based Search

Searching for items or entities by matching terms and their contextual meanings is known as Semantic keyword-based search. As a result of using traditional web search engines, many users are now accustomed to keyword searching. One popular approach is to use Apache Lucene [8] as the core search engine. Lopz et al. [87] merged several data sources to generate heterogeneous data and enabled the user to navigate them using their interface. The data was represented in tabular format [86]. Then, the user was able to perform keyword searches to generate full-text searches on the dataset using Apache Lucene. Results will not include only a dataset matching the keyword; a Lexical chain will be used to identify more relevant results. Therefore, the user could find related datasets by identifying a topic associated with more than one dataset. For example, the user can find a dataset with the same content, such as two entities from two datasets linked using owl:sameAs, or finding datasets with similar entity namings.

Dub-STAR SMS Client [34] also uses Lucene to retrieve traffic data, which was crowd-sourced using social media, based on the user's SMS text messages. This type of search is highly effective and provides high accuracy. However, it is only limited to finding relevant entities rather than constructing complex queries that contain matching relationships and applying filters [51]. Thus, it is suitable for only particular search applications that require basic text searches [28].

6.4 Question Answering

Question Answering systems will interpret the user's question in natural language and return the appropriate answer in the same natural language [65]. The user's primary interest is obtaining the correct answer. Thus, the user is not

[8] https://lucene.apache.org/

expected to understand the underlying structure of data nor the query language [112]. For example, if the user wants to know 'What is the capital of England?', the system will evaluate the question and return 'London' as the answer.

Freitas and Curry [46] introduced a natural language interface using the distributional semantic model to allow the Lay-users to write questions (querying the data) using natural language without the need to learn or understand anything about LD or SPARQL query language. The query processing is vocabulary-independent, which means that the user is not limited to the exact word. Instead, the algorithm compares the query term with all of the relationships (predicates) related to the main subject to weigh up the best match. Thus, for a 'Subject' representing a human, the user could use a word such as "son" in a place of "child" to answer the same question. A quantitative evaluation of the proposed question answering system was made using the "Question Answering over Linked Data 2011" (QALD-1) dataset to evaluate the interface over DBpedia. The results were outstanding, with the system exceeding all the previous systems. Several alternative strategies have been employed to achieve the same objective, including Aqqu [18], which employs templates to determine keywords' relationships, NLQ/A [118], which takes a non-NLP approach and relies on users to resolve linguistic ambiguities, SINA [108], which is a keyword-based natural language interface that utilises Hidden Markov Model to identify relevant resources, CASIA@V2 [**?**], which uses Markov Logic Network to model ambiguities as soft constraints, and BELA, which proposes a multi-layer NLI approach to improve confidence[107].

onIQ [38] is a tool that translates queries in natural language into SPARQL. It uses spaCy library [9] which the author assumes to gain a better performance over the other solutions such as using of Stanford Parser. onIQ was not evaluated, so this assumption needs to be validated. The tool is limited to questions beginning with interrogative words that are also formally structured. Currently, the tool is most suitable for Tech-users, as it expects users to form the queries correctly to generate accurate results.

The RDF Data Cube Vocabulary [10] is used to represent statistical data-cubes which is multi-dimensional values in LD [63]. Answering questions over statistical data using this type of vocabulary has not been appropriately explored [36]. For example, what was the average monthly income for a UK football player in 2019? CubeQA [64], and QA^3 [11] both tried to address this issue. The overall evaluation of both tools shows that QA^3 has a 9% higher F-score. In addition, unlike QA^3, CubeQA does not support SPARQL subqueries.

6.5 SPARQL to Natural Language

Interpreting the meaning or purpose of a specific SPARQL query can pose considerable difficulties for Lay-users. As previously mentioned, it is unreasonable to assume that all users are familiar with SPARQL. Therefore, a more practical approach involves translating queries into the user's native language, thereby facilitating efficient query comprehension [95, 115]. Tools such as SPARQL2NL [94] and SPARTIQULATION [41], which convert SPARQL queries to natural language, can effectively reduce the cognitive load associated with interpreting SPARQL queries, even for experienced users.

SPARQL2NL translates SPARQL queries into natural language based on the chosen SPARQL endpoint, enabling users to execute queries and retrieve results. Subsequently, the tool explains the results in natural language, obviating the need for users to grasp RDF or the connections between resources, as the tool presents explanations in a textual format [94]. Conversely, SPARTIQULATION employs a 'Document Structuring' technique to translate SPARQL queries into natural language by transforming the entire query into a graph representation before rendering it in natural language. However, this approach precludes the possibility of tailoring translations based on data structure variations, as certain messages are hard-coded. Despite the necessity for further refinements, both tools have demonstrated promising outcomes [94, 41].

7 Semantic Web Solutions

This section explores deeper into the current semantic web solutions by conducting a thorough analysis of research papers published as an in-use resource or application. The primary objective of this investigation is to widen the search scope to extract common usage themes and trends in the domain, ultimately contributing to a better understanding of the current state of the field. The following subsections present the identified themes, accompanied by an analysis of each usage theme, highlighting their significance and impact within the field.

[9] https://spacy.io/

[10] https://www.w3.org/TR/vocab-data-cube/

7.1 Virtual Assistants

Conversational AI and Virtual Assistants (VA) provide alternative ways for end-users to interact with the system naturally. Users are not anticipated to grasp any query languages or even apprehend the data structure. Mihindukulasooriya et al. [91] proposed the Dynamic Faceted Search (DFS) system that uses faceted search through Virtual Assistants in the IT technical support domain. Users must begin by conducting a keyword search for their issues. Then, the Virtual Assistant offers a list of options that will guide them through the process of identifying the problem. The list is dynamically generated using a faceted search algorithm by extracting the taxonomy from Wikidata to determine the user intent. The list knowledge induction process is an unsupervised learning approach in which the user is not restricted to any expected input, such as a brand or category. The authors claim that the system is domain-independent as it was also used in different domains.

The simplicity of this proposed Virtual Assistant is noteworthy, as the use of LD to dynamically create user choices lowers the need for expert input to build the system, which speeds up system production. Therefore, the options are unique and relevant to each user's questions. The user experience is completely guided through the faceted search process.

To evaluate the system, the authors used two real datasets. The first dataset was TechQA, a publicly available dataset with 610 question-answer pairs, while the second is a private dataset with 50 question-answer pairs. Both datasets were related to the IT technical support domain. They conducted three types of experiments that include Quantitative, Qualitative and Subject Matter Experts (SMEs) Evaluation. In the quantitative experiment, all matrices outperformed ElasticSearch (the baseline). For the qualitative testing, they used the human-in-the-loop technique to examine randomly chosen queries manually. The results showed that, on average, 60% of the autogenerated options were helpful. However, in the SMEs evaluation, the options relevancy was assessed at only 50%. Two experts have evaluated the options with significant variance in their decisions affecting the final results. Thus, more expert users must be included in the evaluation to lower such variation to improve the overall test quality.

Barisevičius et al. [13] introduced the Babylon Chatbot to provide patients with general health information or to triage them based on the urgency of their health conditions if they needed medical assistance. The authors have created significant LD by integrating data from multiple data sources. Then, they created a KB-Explorer, which is a LD browser for exploring and debugging. The browser also provides text annotation if needed as a debugging feature. However, only Babylon Chatbot was meant for the end-user. The chatbot conversation is guided through the options the patient will receive, which increases the system accuracy, but the user cannot proceed faster by describing their condition in a single message. In terms of evaluation, the given data was insufficient and not clearly explained, though the precision was 0.967 and recall was 0.799, which is encouraging.

Farah et al. [42] created a reasoning engine for a telecom company as a way to review all of the company's historical data regarding device maintenance to suggest a solution to issues, such as rebooting the device or following a specific procedure. The data model was trained using three months of data, which consisted of about 61 million error code records and 3.6 million entries. The data were modelled as a Knowledge Graph to make them semantically available. Then, they created a Voice-based chatbot as the end-user interface. The chatbot was intended to seek to understand the customer's problem and propose a solution. Therefore, the chatbot will iteratively take the customer into a question/answer loop until it reaches a conclusion.

The authors conducted a quantitative experiment with 5,000 problems where the result was only deemed relevant if the correct answer was one of the first five suggestions. The accuracy was 0.58, while the Precision, Recall, and F1 Score were 0.69, 0.58, and 0.60, respectively. They also discovered that the dataset might suffer from missing information affecting the results. Thus, they created a system to recommend alternative solutions to unresolved issues. After a second experiment using the revised system, the Accuracy, Precision, Recall and F1 scores were 0.82, 0.84, 0.82, and 0.83, respectively, showing excellent improvement in results.

The use of Chatbots and Conversational AI offers a natural way of retrieving knowledge. On the other hand, the current usage is solely concerned with diagnosing the issues and reaching a conclusion.

7.2 Microdata and RDFs Authoring Tools

While a knowledge graph must be created and edited by Semantic Web experts, they are not necessarily specialists in the related knowledge domain. Similarly, domain knowledge experts are not necessarily semantic web experts. For example, creating medical ontology requires some involvement from medical field specialists. Therefore, assistive tools must be implemented to motivate these Lay-users to verify and modify their domain knowledge ontology appropriately.

Manually annotating web textual content using Microdata [1] requires high levels of technical knowledge of semantic web annotation. The Seed [40] tool aims to fill the gap between semantic web annotation and mainstream users. The tool does not require any experience with semantic annotation as it will display the web content as a WYSIWYG editor. Users can thus annotate any word or a phrase as a Location, Organization, Person or Other by simply highlighting the target and linking it to the related entity. Additionally, the tool supports data facet browsing either by using the annotated information pane or the entity summary pane. The tool is thus generally intuitive and easy to use. However, it does not support RDF annotation, limiting it to Microdata annotation.

Controlled Crowd-sourcing is an attractive approach that involves the community in ontology curation. Gil et al. [48] introduced Linked Earth Platform (LEP), where users can propose new terms as part of the crowd ontology that, if approved, become part of the core ontology for the LEP. The LEP is, in turn, equipped with a dedicated annotation interface that displays the dataset ontology, including both missing and crowd properties. The platform keeps participants motivated by giving them credit for their contributions, as well as supports supporting community discussions and making decisions by voting. The annotated data can be visualised in a map-based view.

As RDFs are constructed in triples, writing these manually to create a LD resource and all of its corresponding data can be time-consuming. RDFWebEditor4Humanities [79] is an RDF annotating tool that is intended to assist the user by offering suggestions using case-based reasoning, to develop suggestions relevant to the context that are sorted accordingly. The tool has four versions, which share an interface with a variant in the associated suggestions. The interface is distributed into three text fields that express the RDF triples (subject, predicate, and objects). The use of the interface thus requires some Knowledge about RDF and LD.

The evaluation of the RDFWebEditor4Humanities involved human and automatic evaluation. The first experiment required each user to create 10 randomly chosen resources from a list of 30 items. Participants then completed a survey to give feedback about each version, using the Likert scale [3] to rate the suggestion relevance, where 0 was the lowest and 7 was the highest. The overall results for the Basic, Deductive, Cased-based, and Combination editors were 3.4, 5.7, 5.3 and 7, respectively. The first experiment proved that the Cased-based editor did not improve the suggestions while the Combination editor was influential. The second experiment offered performance analysis based on analysing the suggestion list and comparing it to the expected value to assign a ranking. The lower the rank, the better the suggestion. By far, the Case-based editor and Combination editor performed much better. Both versions have achieved almost the same results.

Schema.org has become the standard schema to create ontologies for LD. Schema.org contains a generic vocabulary covering various domains, including numerous unrelated vocabularies. Customising Schema.org to precisely fit a specific domain is challenging as it requires the removal of unrelated properties and types. In addition, it requires defining the local properties and constraints. The Domain Specification Editor [120] is a tool intended to help the user to generate domain-specific custom annotation. The tool allows the user to manually create the annotations by selecting a 'Domain Specification' from the list or automatically generates the annotation by fetching a web page URL. The tool is thus only meant for Experts to assist in building and validating custom ontologies that follow Schema.org specifications.

The evaluation of the Domain Specification Editor included a usability study and usage survey. A System Usability Scale (SUS) [12] was used with Likert scale [3] response categories across 37 participants, distributed as eight experienced users and 29 mainstream users. In general, the usability results were 'good', with 75% of the experienced users finding the tool to be excellent. However, the inexperienced users found using the tool more difficult, with only 20% rating it as excellent. The second experiment involved 14 participants with experience in creating annotation with schema.org. They were asked to create annotations with and without the tool domain-specific patterns. Then, they had to answer some related questions. The results showed that 78.6% of the participants found the domain-specific patterns to be simple to use. Moreover, all participants reported that it was helpful. Half of the participants stated that it facilitated the process, saved their time, and assisted them in discovering new properties.

Another successful annotator is Smart Topic Miner (STM) [97], an automated classification tool used to support the Springer Nature editors. STM parses publications metadata to display the annotated topics alongside relevant papers, classification labels, and useful analytical information about such papers within the proceedings book. The editor can then decide whether or not to check and modify any annotations manually before submitting them. The use of STM enhances the general findability and accessibility of the resources. The STM has shown promising results in terms of usability, with a SUS score of 76.6. However, it is only limited to the computer science field. Also, STM can only analyse the metadata rather than analysing full-text publications. In addition, the inferences behind STM suggestions are hidden, leaving no clear verification approach to build user trust.

Smart Topic Minor 2 (STM2) [105] is the second version of STM. This version addresses some of the first version issues based on the editors' feedback. The search approach has been modified to offer the user an explanation for the

tool suggestions. Besides, the UI has been overhauled to be more dynamic. In addition, STM2 also considers the historical data for the earlier editions of the same conference proceedings. As STM2 is integrated with the Computer Science Ontology Portal [104], it is still restricted to Computer Science research. STM2 SUS score was 93, which is an excellent score. This evaluation shows a dramatic change in terms of usability compared to the old version.

CodeOntology [8] is a tool that enables the parsing of Java source code to generate RDF triples. It also links the code comments with suitable DBpedia resources. The idea is to allow the user to query the source code through the produced ontology using SPARQL. Thus, the software elements become semantically accessible. The authors have also deployed a QA system called AskCO [10] to examine the source code querying for the generated ontology. AskCO thus handles the user queries in natural language and interprets them by calling the most suitable function. However, CodeOntology is limited to use on Java code.

7.3 Mobile Applications

Nutrition and maintaining a healthy diet is one of the LD applications. Dragoni et al. [39] introduced PerKApp, which is a mobile application to monitor the user's diet and activity to promote healthier lifestyles. The application tries to send persuasive messages to the user based on their collected data. The application uses Rule-Based Reasoning that was placed by experts such as physicians and dietitians to detect any violations of user behaviour (i.e., not consuming the right amount of calories on breakfast). User data are inserted into the LD and validated against the rules immediately, with daily and weekly validation to detect any missed violations.

Donadello and Dragon [37] extended the work on PerKApp by empowering LD with AI and linking it with the user's personal health records to monitor the diet habits and predict any nutritional diseases that might be linked with the user's diet. Users can use the mobile application to track their diet by taking pictures of their food. Then, the system will classify the image to identify the food category. The application is the only means for the end-user to connect to the LOD. The end-user food consumption habits are then visualised as a progress bar to promote healthy food consumption. Generally, the application is intuitive and provides higher data abstraction with no direct engagement with the LD.

Dragoni et al. [39] generated a performance analysis based on real data collected from 49 users who used the application regularly for 45 days. The results showed that the daily reasoning time on average was 1 second, as compared to the weekly reasoning time of 14 seconds on average. Thus, the reasoner time was also correlated with the number of violations. An SUS questionnaire was then used to examine the experts' opinions on the usability of the setup process and to define rules. The average score was 81.5, which is excellent as per Bangor et al. [12] proposed scale.

Additionally, Donadello and Dragon [37] conducted a quantitative experiment to examine their classifier and two qualitative experiments to judge the usefulness of their mobile application. When tested, their initial single-based classifier failed to classify some food content creating a domino effect that damaged the system's suggestions and undermined its primary purpose. Therefore, they switched to multi-label classification using a new labelled dataset. As a result, the classification was significantly improved. In addition, the usability test result for their application was 83, which is rated as excellent. Then, to assess the impact of the mobile application's motivational messages, users were split into two groups, with 92 in the first group receiving such messages and 28 in the second group (control group) receiving no messages. The results showed that users from the first group were less likely to violate the diet guidance.

7.4 Entity Centric Dashboards

Leskinen et al. [82] presented the Actor Ontology from a Finland Finnish World War II dataset in an intense Spatio-temporal model in the WarSampo portal [11]. The data was shown from four main viewpoints: Persons, Military Units, Articles and Photographs. The first perspective reflects "Person" information by allowing the user to search for a person's name and thus access all of the information linked to that entity. Similarly, users can use the second viewpoint to search for the Military Units, listing all the information related to the given unit, such as photos, personnel and battles, including a map displaying the unit location with an interactive timeline. The third viewpoint is an archive of "Kansa Taisteli Magazine" articles that allows users to filter articles using information such as author name and issue date. The final viewpoint is of photographs organised as an image gallery, where the user can search for an image by specifying the period, location, military unit, and photographer. These web interfaces were not evaluated [82].

Different viewpoints may be connected by the shared entities. For example, when exploring a person, the user can navigate to the person's military unit and then explore related images. This facilitates smooth data browsing by allowing users to begin with any desired viewpoint to find the required data. However, the web interface mainly focuses on

[11] https://www.sotasampo.fi/en/

'Faceted Search' and thus does not support the inclusion of two or more unrelated entities in searches. For example, users can not find articles written by either author A or author B in a single search.

One of the useful applications of semantic web technologies is knowledge extraction as used to track illegal activities. Kejriwal and Szekely [74] created an intelligent search engine to assist investigators of Human Trafficking. The engine uses NLP and Information Retrieval techniques to extract knowledge from millions of web posts to build the Domain-specific Insight Graph (DIG). A DIG GUI is then used to pass the query to the engine and display the results. The investigator must enter the required search terms, and the results are then displayed in a ranked entity list, sorted in descending order. Thus, the user can browse each entity on an entity-centric visualisation or narrow down the results using the faceted search options.

The evaluation of [74] was not published due to the confidentiality of the data. However, the authors shared the techniques alongside some of the results. They have evaluated their GUI by performing controlled usability testing on 8 SMEs. The participants came from four different states in the USA, and each participant was asked to spend two hours daily for one week answering eight 'lead generations' and eight 'lead investigation' questions, in addition to a 45-minute training session. They used System Usability Scale (SUS) questionnaire to measure their GUI usability and scored above 70, which is recognised as above average.

Fernàndez-Cañellas et al. [43] created a system to extract the current media news from various sources and languages with the aim of identifying trends and similar articles. The user can then access these findings through a web dashboard that presents the revealed topic alongside the related "when, where, and who" answers, as well as displaying all media that mention the same topic, with related tags linking the topic to other related topics. The interface is simple and well structured from a journalistic perspective.

The system has two main components, News Event Detection and Dynamic Entity Linking. The first is responsible for discovering and classifying topics so that similar topics can be identified and grouped, while the second models the semantic relations between the events. Keywords can then be detected and linked to the related entities allowing the user to identify all events related to a specific tag or person. The authors also conducted a quantitative evaluation for each component separately to compute the 'Classification Evaluation Metrics' such as accuracy and precision in order to identify any deficiencies within each system component; however, the dashboard was not evaluated.

7.5 Web APIs

Web developers primarily rely on Web Application Programming Interfaces (APIs) to consume Web Services and retrieve data [116]. Each API act as a translator between two applications, allowing them to communicate. Therefore, for a web developer to connect a web application to the LD, they can either use the relevant LD SPARQL-endpoints or create a specific Web API [90]. The first option involves sending SPARQL queries to the endpoint and retrieving data as RDF triples in a format such as JSON, CSV or as supported by that API. The data is then analysed and presented in HTML format. This approach requires the developer to have experience with SPARQL, RDF, and Knowledge Graphs. The second option is to use a Web API by requesting a distinct URL that represents a resource or a service, possibly, with predefined parameters, with the data received as an HTTP response. The Web API should either have the OpenAPI Specification (OAS)[12] description, which offers documentation for both humans and machines or at least has the traditional documentation so the client knows which services are offered by that API. The format of the transferred data can be a Web-friendly format such as JSON, JSON-LD [13], or XML. With the second option, the developer must be familiar with the technology and be able to access LD in a straightforward approach.

However, creating a Web API requires regular maintenance. Web APIs lack standardisation and act in a "Blackbox" manner, keeping all the querying behind the scenes [116, 90]. Therefore, Meroño-Peñuela and Hoekstra [90] introduced grlc [14], a tool that dynamically generates well-standardised RESTful APIs by transforming SPARQL queries. The user must store the SPARQL queries in a GitHub repository, and the tool then uses these as the source for constructing the API. This allows web developers without experience in SPARQL to efficiently access LD. Despite that, grlc does not support customising the API results, as it returns only these formats support by the endpoint. Thus, consuming the LD results may still be a challenge.

Lisena et al. [83] addressed this issue by introducing SPARQL Transformer, a tool that allows users to write SPARQL queries in JSON in the form they wish to see the final result. The tool accepts SPARQL queries as plain JSON or JSON-LD and then returns the results with a matching JSON structure. The most remarkable feature of this tool is

[12] https://www.openapis.org

[13] https://json-ld.org/

[14] https://grlc.io/

that it can be integrated with grlc to allow web developers to customise their API output. SPARQL Transformer lacks expressivity and is currently limited to SELECT queries.

Alternatively, Garijo and Osorio [47] have introduced an Ontology-Based APIs (OBA) framework that extends the work done by tools like grlc, allowing automatic creation of an OAS from the selected ontology. Then, generate APIs to access the LD that follows the same OAS, with the returned results in a JSON format that matches the OAS. Thus, the involvement of the knowledge graph experts is reduced, and fewer efforts are required to process the results. In addition to supporting SELECT expressions in SPARQL, OBA supports INSERT, UPDATE, DELETE, and CONSTRUCT expressions. In terms of limitations, OBA is affected by ontology modification, which will produce a new API version. Also, enormous ontologies will also create massive APIs that may be slow to access.

The evaluation of grlc included a qualitative evaluation by presenting the users' feedback on in-use projects. A quantitative evaluation was also conducted to test the speed and performance of the tool. As expected, the performance of grlc was almost constant, as it introduces a steady overhead between the application and the endpoint, so its performance is not affected by the size of the dataset. During the experiments, response time never exceeded 187.9 ms, which was deemed adequate.

In addition, the SPARQL Transformer was quantitatively evaluated to test its performance as a stand-alone tool without integration with grlc. The results showed that using SPARQL Transformer was slightly slower than direct SPARQL endpoint querying, but that this delay was less than 100 ms, which was seen as acceptable. A questionnaire was also conducted to reflect user opinions regarding the data representation and the JSON format in comparison to the direct endpoint querying. The results suggested that users generally prefer the tool but that their decisions were not affected by the level of data nesting, despite the authors' assumptions.

The authors of OBA also implemented performance analysis to measure the time taken to render the results into JSON and to assess its performance across a different range of requests. The first analysis showed that overall, OBA had overhead that averaged below 150ms and never exceeded 200ms. In the second experiment, when using proxy caching, the 60 queries per second were handled in less than 200ms. However, without proxy caching, the performance dramatically diminished in cases where ten requests or more were received, exceeding 5 seconds, which was considerably worse.

8 Evaluation and Research Validation

8.1 User Study

Evaluation is an essential part of any proposed tool or interface, as it is needed to determine the usefulness of the introduced design or technique. Part of the reviewing process of this paper thus dedicated to exploring the evaluation design of each paper. Table 4 summarises the evaluation techniques and tools used by each paper. The evaluation methodologies ranged across quantitative, qualitative, and mixed methods [50]. According to the ISO 9241-11, the quantitative usability evaluation has to achieve satisfaction, efficiency and effectiveness with measurable characteristics [22]. Most of the reviewed work used a System Usability Scale (SUS) questionnaire to measure user satisfaction, with this score interpreted using a Likert Scale [3]. The SUS score was occasionally normalised using the Percentile Rank [15] to give a percentage or a grade. Likert Scale was also integrated with various surveys to allow comparison of two or more platforms based on users' opinions.

The NASA-Task Load Index (NASA-TLX) [59] is another evaluation tool that distinguishes between multiple platforms using the user-perceived workload. As with SUS, NASA-TLX is used to measure user satisfaction, but the latter is distributed to evaluate workload in a manner that includes frustration, difficulty, mental effort, physical effort, temporal effort, and performance, so that lower scores are better.

The effectiveness of the system was usually measured by means of user ability to complete the required task successfully. In addition, Subject Matter Experts (SMEs) were used to verify output to determine system accuracy. Alternatively, Kuric et al. [78] have employed the number of hints which was given to the user to accomplish the task as a method of evaluating the accuracy. Efficiency was usually determined by the amount of time the user spends to complete the task, though performance metrics, such as f-measure, recall, and precision, were also used to measure the efficiency of the information retrieval process.

The qualitative evaluation methods combined a variety of tools to capture user observations and feedback. The most common approach was to create a controlled experiment, followed by open questions about the system. Usually, these open questions were given as part of the questionnaire completed by participants. However, sometimes they took the

[15] https://measuringu.com/interpret-sus-score/

form of an interview. These questions are typically preceded by assessment questions targeted at user background details such as the user years of experience. The "think-aloud protocol" was also used to collect the user observations while using the system, followed by a debrief session to verify their input.

In terms of participant quantity, numbers ranged from 1 to 120 participants. The most frequent sample sizes were 6, 8, 10, 14, 15, 48, and 120 participants. According to Nielsen and Landauer [96], the optimal sample sizes for small and medium-large projects are 7 and 15, respectively.

8.2 SPARQL Query Evaluation

The majority of the reviewed tools focused on the user study to demonstrate their usefulness, while the correctness and validity of the generated queries were not explicitly stated. Most of the Question Answering tools relied on QALD challenge to measure performance metrics, such as precision and recall rather than question matching the SPARQL query [99]. Other reviewed work appeared to depend on users offering satisfaction feedback about the queries.

Evaluating tools by matching the autogenerated SPARQL query with the ideal SPARQL query is known as the "black-box" approach [30]. The black-box approach examines only the system input and output to evaluate the system [93]. The idea is that, as two equivalent queries may be structured differently, these queries should be matched using the produced results [30, 6]. If the LD is periodically updated, an exact match for the output is impossible, as even the

Table 4: Evaluation techniques used by each paper.

		Quantitative								Qualitative			
Tool	Sample Size	SUS	Likert Scale	NASA-TLX	SME User Evaluation	Task Completion	Performance Metrics	Top-N Metrics	Time	Think-aloud	Evaluators' Assessment	Open Questions	HITL
Mihindukulasooriya et al. [91]	2 SME				✓			✓					✓
Barisevičius et al. [13]	N/A						✓						
Farah et al. [42]	N/A						✓						
Donadello and Dragon [37]	120	✓					✓				✓		
Dragoni et al. [39]	49	✓							✓				
Fernàndez-Cañellas et al. [43]	N/A						✓						
Lisena et al. [83]	55								✓		✓	✓	
Meroño-Peñuela and and Hoekstra [90]	6								✓			✓	
Garijo and Osorio [47]	N/A								✓				
Kejriwal and Szekely [74]	8 SME	✓			✓						✓	✓	
Lasolle et al. [79]	1 SME		✓		✓		✓						
Şimşek et al. [120]	51	✓	✓								✓	✓	
Eldesouky et al. [40]	120	✓				✓	✓		✓		✓		
Osborne et al. [97]	8 SME	✓			✓						✓	✓	
Thanapalasingam et al. [111]	14 SME	✓			✓						✓	✓	
Salatino et al. [105]	9 SME	✓			✓		✓				✓	✓	
Sorce et al. [**?**]	21	✓	✓	✓							✓	✓	
Vega-Gorgojo et al. [114]	15		✓			✓	✓		✓				
Kuric et al. [78]	15	✓				✓			✓	✓	✓		
Haag et al. [54]	10									✓	✓	✓	
Hoefler et al. [62]	14			✓		✓				✓	✓	✓	
Haag et al. [57]	6								✓		✓	✓	
Vargas et al. [113]	28		✓	✓		✓			✓		✓	✓	
Soylu et al. [110]	10		✓			✓			✓	✓	✓		
Kaufmann et al. [72, 73]	48	✓	✓			✓			✓		✓	✓	
Zenz et al. [117]	-					✓			✓				
Ferré [44]	26	✓				✓			✓		✓	✓	

same query will produce different results. A "grey-box" approach is thus required, in which the query is semantically evaluated by checking for the existence of specific triples with the help of domain experts [30, 6]. However, the majority of the reviewed tools did not specify whether or not the resulting queries was examined as part of their evaluation.

It is difficult to evaluate these tools independently in terms of the generated queries, as the systems are generally inaccessible or configured to query only a specific SPARQL endpoint. For example, out of the reviewed Form-based QBs, only three allow the user to query Wikidata, while one does not support exposing the SPARQL query, making the evaluation of generated queries infeasible.

9 Findings and Discussion

This survey comprehensively examined a diverse array of tools designed for accessing LD, with an emphasis on those that generate SPARQL queries, as the W3C endorses SPARQL as the recommended query language for LD. The majority of these tools cater to a wide range of users; however, they often exhibit shortcomings in accommodating specific user categories, particularly Lay-users. While experienced users, well-versed in RDF, can readily comprehend most interface metaphors, Lay-users frequently grapple with these tools due to the requirement of prior RDF knowledge [33, 35, 113, 2, 62, 78, 46]. Figure 9 briefly summarises the main advantages and disadvantages of the primary types of SPARQL query builders examined in this survey.

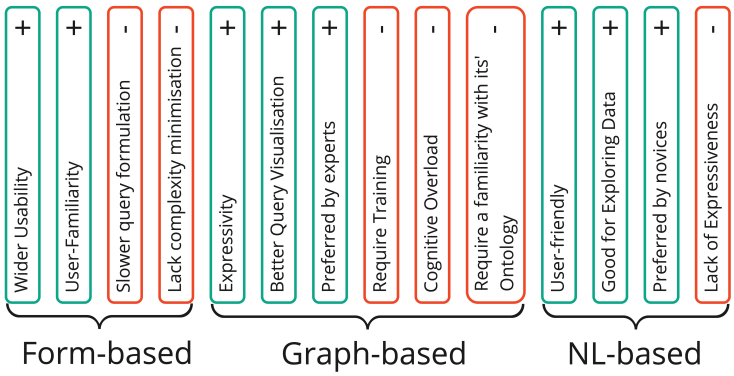

Figure 9: The summarisation of the findings by listing the Pros (+) and Cons (-) of the main types of SPARQL query builders.

User-Friendly The NL-Based Query Builders are most user-friendly interfaces, as these offer greater levels of abstraction than other tools, and thus do not require the user to have any previous knowledge about the LD or SPARQL. These tools are meant for Lay-users and thus aim to remove the barriers to accessing LD. As a result, however, some expressiveness in terms of constructing the query is lost. QUICK thus adds an additional layer to recognise user intention to improve accuracy, while SPARKLIS eliminates arbitrary keyword searches by limiting sentence contraction to the available option boxes. However, constructing complex queries takes more time with these tools, making them more suitable for exploring data content and creating simple queries.

Contrarily, Several tools convert SPARQL queries and their retrieved results into a more human-readable formats, and these tools explain in detail what a particular SPARQL query does. In addition, they can convert seemingly meaningless URI into something more readable. However, they do not support converting such natural language texts back into SPARQL.

Wider Usability Form-based Query Builders are suitable for a wider variety of users, as well as allowing more detailed querying of the data region of interest. Some of these interfaces rely on the structure of the RDF triple pattern to reduce data abstraction, though most such tools offer an auto-filling option to speed up the learning curve, as seen in WQS and VizQuery. To take advantage of users' familiarity with spreadsheets, tools such as LDQW, ExConQuer, and Falcons Explorer show their initial results in a tabular format, allowing users to shape the retrieved results using filters to match their specific needs. Both Experts and Lay-users thus benefit from using these types of query builders.

Visualisation and Detailed Querying Graph-based Query Builders require tremendous cognitive load and prior knowledge of SPARQL, making them unsuitable for Lay-users. These query builders excel in visualising queries to reflect their LD structures, allowing the full expressiveness of SPARQL to be displayed, and thus even experts

may not be sufficiently familiar with the data structure. Some tools have tried to overcome this unfamiliarity issue by offering a toolbox with a fixed set of draggable objects such as ViziQuer, QueryVOWL, OptiqueVQS, and RDF Explorer. However, for non-experts, constructing the query path remains difficult, based on a need for understanding of the connections between objects.

10 Research Challenges and Future Directions

While the reviewed works and solutions cover a broad range of domains, several unaddressed gaps and areas for improvement remain. Moreover, the usability of several researched tools has not been assessed or tested, warranting further exploration into the practicality of the proposed solutions. With regard to broader usability, none of the examined tools facilitates the integration of basic and advanced user interfaces to minimise barriers for Lay-users, which could expedite their adaptation to the interface while still accommodating advanced querying for experts. Furthermore, NL-based QBs do not support the conversion of SPARQL to NL. For instance, tools such as SPARQL2NL [94] enable users to convert SPARQL to NL, a feature not offered by NL-based QBs. Incorporating this functionality could assist Lay-users in evaluating the generated SPARQL. Therefore, it is essential to investigate this aspect to determine whether the inclusion of such capabilities might improve the evaluation results of these tools.

Supporting Web of Things (WoT) and Spatial Data The advent of smart buildings and cities, in addition to the expansion of the Web of Things (WoT) domain, has introduced novel concepts into traditional LD, including sensors, observations, and spatial data. The majority of reviewed query formulation tools primarily focus on extracting general knowledge from conventional LD. Although these tools are designed to operate with general LOD, they lack basic concepts for accessing and exploring LD adhering to specifications such as the "Sensor, Observation, Sample, and Actuator (SOSA)" ontology [69]. Reapplying some of the reviewed techniques or considering novel visualisation approaches for this type of LD is crucial for end-users.

In contrast to traditional LD, which may involve simple RDF patterns like 'Adam is Human', information retrieval in the context of Smart buildings differs significantly. For instance, a thermal sensor placed in an area with multiple sensors could collect readings on various measurements at regular intervals. Consequently, the sensor's readings would reflect a many-to-one relationship, linking multiple records to a single sensor. Although these relationships are described in the ontology, their visualisation and connections are more complex and condensed.

In terms of querying data to obtain information related to the building's condition, the sensor itself is of greater interest than individual readings. Users typically seek the ability to query all data associated with a single sensor, filtering out irrelevant readings by matching a specific pattern rather than focusing on a single reading corresponding to a simple RDF pattern.

Results Perception During Query Construction Form-based QBs appear to be the least intimidating interfaces for novice users among other QBs; however, usability enhancements are needed. The FILTER/FLOW MODEL [55] demonstrates that providing preliminary results indicators during query construction aids users in formulating accurate queries. QueryVOWL [56] has also implemented a similar feature by assigning a numerical value to each node to represent the size of the current query results. This approach provides users with a visual indication of the number of results corresponding to each node, helping them assess the effectiveness of their queries during the construction process. None of the other query builders supports such a feature, which involves displaying potential results during query construction, warranting further investigation of this technique.

This is important due to the fact that, as we previously discussed, even experts may experience ambiguity regarding expected results, which can impact the query construction process. Consequently, offering users insight into possible outcomes during construction could enhance their overall experience. For instance, visual indicators such as increasing the thickness of the visual model to represent the size of the results can help users assess the effectiveness of applied filters. Alternatively, employing darker colours to indicate data density could also convey the intensity of the results. If the results incorporate geospatial data, maps could serve as an effective visualisation method, significantly influencing user perception. Additionally, when users visually select an entity, presenting all associated entities could improve the query construction process compared to users having to infer these entities. Therefore, the perception of results constitutes a critical factor that can positively impact usability.

Avoiding Non-empty Results Facet search tools, commonly employed in LD browsers, are typically designed to prevent users from encountering empty results by exclusively displaying relevant resources. However, as shown in Table 1, 2, and 3, the majority of query formulation tools do not incorporate such a feature. Consequently, users may unintentionally construct queries that only retrieve empty results. While some reviewed tools attempt to mitigate this issue by providing examples, this approach is insufficient to eliminate the problem entirely. One possible implementation

to address this concern is to disable arbitrary inputs and remove options that would yield empty results. Therefore, to enhance query construction and improve usability for Lay-users, eliminating empty results could increase user confidence and satisfaction.

Effective Entity Recognition and Discovery All reviewed query builders that incorporate text search functionality rely solely on simple keyword searches to identify entities by their URI or associated labels and comments. For example, a basic search may fail to retrieve relevant entities if a user types 'Car' while the resource description contains the terms 'Automobile' or 'Vehicle'. In addition, the inference is a powerful feature of LD; however, as shown in Table 1, 2, and 3, none of the tools support inferences. An instance of inference includes the ability to determine an equivalent URI using owl:sameAs or to identify indirect relationships, such as 'John lives-in London', 'London is-in England', which implies 'John lives-in England' is valid. By integrating inferences with advanced NLP techniques to augment the basic text search, query construction could be significantly improved.

Supporting Conversational AI NL-based and Keyword-based approaches focus on extracting knowledge from LD using sentences or keywords. However, these methods lack the ability to determine user intent and, as a result, cannot provide feedback to improve query construction. In contrast, a Virtual Assistant (VA) represents an End-User Development (EUD) tool not yet explored in the context of accessing LD. Barricelli et al. [15] discussed the potential of employing VA and Conversational AI in the IoT domain to facilitate end-users' management of their IoT environments.

VAs can more accurately capture user intent and present results in the user's natural language. Current VA implementations are limited to deriving conclusions or utilising predefined SPARQL templates, necessitating query modifications by replacing entity values, as demonstrated in Mishra et al. [92]. However, no support for accessing data using adaptive or scalable approaches has been identified thus far. Exploring such approaches could offer significant advancements in accessing and querying LD through VAs.

Improving Usability by Integrating Multiple Tools Each of the reviewed tools and visualisation approaches has its advantages and disadvantages. Nevertheless, these tools have not explored the effectiveness of combining multiple approaches to address known limitations and enhance tool usability. For instance, it remains unclear whether merging form-based tools with graph-based tools would reduce the cognitive load observed in purely graph-based instances while improving expressivity, which is constrained in the form-based approach. Similarly, combining NL-based with a form-based approach might enhance the speed of query construction.

The integration of multiple visualisation paradigms should be cautiously examined to prevent overwhelming the user by increasing the tool's complexity. This implies that additional functionality should be introduced as an assistive tool or an alternative visualisation that is closely linked to represent the same query. By carefully exploring such combinations, researchers may develop more effective and user-friendly solutions for querying LD.

11 Conclusion

The Linked-Data (LD) paradigm has demonstrated remarkable potential for representing information and delivering human and machine-readable formats. However, its potential is constrained by the fact that its users are expected to have a prior understanding of the complex SPARQL query language and the associated concepts to access the data. This paper has reviewed various tools and approaches for accessing and searching LD. The primary focus of the review is on SPARQL Query Builders, which are LD searching tools. The study outlined 22 usability criteria to evaluate these query builders and identify their strengths and weaknesses. The study also provides an analysis of alternative techniques for accessing LD. We have categorised all of the reviewed tools and techniques based on their UI querying approach.

The paper also investigated the current in-use solutions and identified the current usage patterns to identify potential areas for improvement to enhance the search process. Additionally, the study examines the common research validation techniques used in the reviewed papers and highlights some limitations in their validation techniques. The findings reveal that there are weaknesses in the current solutions, particularly in terms of usability, especially for Lay-users. Furthermore, there are some techniques and domains that need to be adequately addressed. The objective is to provide insights into the development of effective and efficient searching tools that can lead to broader adoption of LD and semantic web technologies in a range of new domains.

References

[1] HTML Standard - Microdata, 2022. URL `https://html.spec.whatwg.org/multipage/microdata.html`.

[2] K. Affolter, K. Stockinger, and A. Bernstein. A comparative survey of recent natural language interfaces for databases. *VLDB Journal*, 28(5):793–819, 2019. ISSN 0949877X. doi: 10.1007/s00778-019-00567-8.

[3] I. E. Allen and C. A. Seaman. Likert scales and data analyses. *Quality Progress*, 40(7):64–65, 2007. ISSN 0033524X.

[4] O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: A visual query builder for SPARQL on the social semantic desktop. *CEUR Workshop Proceedings*, 565, 2010. ISSN 16130073.

[5] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, 50(5), sep 2017. ISSN 0360-0300. doi: 10.1145/3104031.

[6] T. Asakura, J.-D. Kim, Y. Yamamoto, Y. Tateisi, and T. Takagi. A Quantitative Evaluation of Natural Language Question Interpretation for Question Answering Systems. In R. Ichise, F. Lecue, T. Kawamura, D. Zhao, S. Muggleton, and K. Kozaki, editors, *Semantic Technology*, pages 215–231, Cham, 2018. Springer International Publishing. ISBN 978-3-030-04284-4.

[7] J. Attard, F. Orlandi, and S. Auer. ExConQuer: Lowering barriers to RDF and Linked Data re-use. *Semantic Web*, 9(2):241–255, 2018. ISSN 22104968. doi: 10.3233/SW-170260.

[8] M. Atzeni and M. Atzori. CodeOntology: RDF-ization of Source Code. In C. D'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, editors, *The Semantic Web – ISWC 2017*, pages 20–28, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68204-4. doi: 10.1007/978-3-319-68204-4_2.

[9] M. Atzeni and M. Atzori. Towards semantic approaches for general-purpose end-user development. In *Proceedings - 2nd IEEE International Conference on Robotic Computing, IRC 2018*, volume 2018-Janua, pages 369–376. IEEE, 2018. ISBN 9781538646519. doi: 10.1109/IRC.2018.00077.

[10] M. Atzeni and M. Atzori. AskCO: A multi-language and extensible smart virtual assistant. In *Proceedings - IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019*, pages 111–112, 2019. ISBN 9781728114880. doi: 10.1109/AIKE.2019.00028.

[11] M. Atzori, G. M. Mazzeo, and C. Zaniolo. QA 3: A natural language approach to question answering over RDF data cubes. *Semantic Web*, 10(3):587–604, 2019. ISSN 22104968. doi: 10.3233/SW-180328.

[12] A. Bangor, P. Kortum, and J. Miller. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *J. Usability Studies*, 4(3):114–123, may 2009. ISSN 1931-3357.

[13] G. Barisevičius, M. Coste, D. Geleta, D. Juric, M. Khodadadi, G. Stoilos, and I. Zaihrayeu. Supporting Digital Healthcare Services Using Semantic Web Technologies. In M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. D'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, K. Thirunarayan, and S. Staab, editors, *Lecture Notes in Computer Science*, volume 9366 of *Lecture Notes in Computer Science*, pages 291–306, Cham, 2018. Springer International Publishing. ISBN 9783030006686. doi: 10.1007/978-3-030-00668-6_18.

[14] B. R. Barricelli and S. Valtolina. A visual language and interactive system for end-user development of internet of things ecosystems. In *Journal of Visual Languages & Computing*, volume 40, pages 1–19. Elsevier Ltd, 2017. doi: 10.1016/j.jvlc.2017.01.004.

[15] B. R. Barricelli, E. Casiraghi, and S. Valtolina. Virtual Assistants for End-User Development in the Internet of Things. In A. Malizia, S. Valtolina, A. Morch, A. Serrano, and A. Stratton, editors, *End-User Development*, pages 209–216, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24781-2. doi: 10.1007/978-3-030-24781-2_17.

[16] S. D. Bartolomeo, G. Pepe, V. Santarelli, and D. F. Savo. Sparqling: Painlessly drawing SPARQL queries over GRAPHOL ontologies. *CEUR Workshop Proceedings*, 2187(January 2018):70–77, 2018. ISSN 16130073.

[17] G. Barzdins, S. Rikacovs, and M. Zviedris. Graphical query language as SPARQL frontend. In *Local Proceedings of 13th East-European Conference (ADBIS 2009)*, pages 93–107, 2009.

[18] H. Bast and E. Haussmann. More accurate question answering on freebase. *International Conference on Information and Knowledge Management, Proceedings*, 19-23-Oct-:1431–1440, 2015. doi: 10.1145/2806416.2806472.

[19] P. Bellini, P. Nesi, and A. Venturi. Linked open graph: Browsing multiple SPARQL entry points to build your own LOD views. *Journal of Visual Languages & Computing*, 25(6):703–716, 2014. ISSN 1045926X. doi: 10.1016/j.jvlc.2014.10.003.

[20] T. Berners-Lee. Linked Data, 2006. URL `https://www.w3.org/DesignIssues/LinkedData.html`.

[21] T. Berners-lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *The 3rd International Semantic Web User Interaction Workshop (SWUI06)*, page 16, 2006.

[22] N. Bevan, J. Carter, and S. Harker. ISO 9241-11 Revised: What Have We Learnt About Usability Since 1998? In M. Kurosu, editor, *Human-Computer Interaction: Design and Evaluation*, pages 143–151, Cham, 2015. Springer International Publishing. ISBN 978-3-319-20901-2.

[23] N. Bikakis and T. Sellis. Exploration and visualization in the web of big linked data: A survey of the state of the art. *CEUR Workshop Proceedings*, 1558, 2016. ISSN 16130073.

[24] J. Borsje and H. Embregts. Graphical Query Composition and Natural Language. *Processing in an RDF Visualization Interface*, 2006.

[25] J. Brunetti, R. García, and S. Auer. From Overview to Facets and Pivoting for Interactive Exploration of Semantic Web Data. *International journal on Semantic Web and information systems*, 9:1–20, 2013. doi: 10.4018/jswis.2013010101.

[26] G. Burel, A. E. Cano, and V. Lanfranchi. Ozone browser: Augmenting the web with semantic overlays. *CEUR Workshop Proceedings*, 449:3–4, 2009. ISSN 16130073.

[27] K. Cerans, J. Ovcinnikova, and M. Zviedris. SPARQL Aggregate Queries Made Easy with Diagrammatic Query Language ViziQuer. In *ISWC 2015 Posters & Demonstrations Track, CEUR Workshop Proceedings*, volume 1486, pages 1–4, 2015.

[28] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L. D. Ibáñez, E. Kacprzak, and P. Groth. Dataset search: a survey. *VLDB Journal*, 29(1):251–272, 2020. ISSN 0949877X. doi: 10.1007/s00778-019-00564-x.

[29] G. Cheng, H. Wu, S. Gong, W. Ge, and Y. Qu. Falcons Explorer: Tabular and Relational End-user Programming for the Web of Data. *Semantic Web Challenge 2010*, (c), 2010.

[30] K. B. Cohen and J.-D. Kim. Evaluation of SPARQL query generation from natural language questions. *Proceedings of the conference. Association for Computational Linguistics. Meeting*, 2013:3–7, sep 2013. ISSN 0736-587X.

[31] M. Console, D. Lembo, V. Santarelli, and D. F. Savo. Graphol: Ontology representation through diagrams. In *CEUR Workshop Proceedings*, volume 1193, pages 483–495, 2014. doi: 10.13140/2.1.3838.3363.

[32] A. S. Dadzie and E. Pietriga. Visualisation of Linked Data - Reprise. *Semantic Web*, 8(1):1–21, 2017. ISSN 22104968. doi: 10.3233/SW-160249.

[33] A.-S. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *International Journal on Semantic Web and Information Systems*, 2(2):89–124, 2011. ISSN 15700844. doi: 10.3233/SW-2011-0037.

[34] E. M. Daly, F. Lecue, and V. Bicer. Westland row why so slow? Fusing social media and linked data sources for understanding real-time traffic conditions. *International Conference on Intelligent User Interfaces, Proceedings IUI*, pages 203–212, 2013. doi: 10.1145/2449396.2449423.

[35] A. De Santo and A. Holzer. Interacting with Linked Data: A Survey from the SIGCHI Perspective. pages 1–12, 2020. doi: 10.1145/3334480.3382909.

[36] D. Diefenbach, V. Lopez, K. Singh, and P. Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems*, 55(3):529–569, 2018. ISSN 02193116. doi: 10.1007/s10115-017-1100-y.

[37] I. Donadello and M. Dragoni. An End-to-End Semantic Platform for Nutritional Diseases Management. In C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, editors, *The Semantic Web – ISWC 2019*, pages 363–381, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30796-7.

[38] I. C. Dorobăţ and V. Posea. onIQ: An Ontology-Independent Natural Language Interface for Building SPARQL Queries. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 139–144, 2020. doi: 10.1109/ICCP51029.2020.9266272.

[39] M. Dragoni, M. Rospocher, T. Bailoni, R. Maimone, and C. Eccher. Semantic Technologies for Healthy Lifestyle Monitoring. In D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, editors, *The Semantic Web – ISWC 2018*, pages 307–324, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00668-6.

[40] B. Eldesouky, M. Bakry, H. Maus, and A. Dengel. Seed, an End-User Text Composition Tool for the Semantic Web. In P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, editors, *The Semantic Web – ISWC 2016*, pages 218–233, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46523-4. doi: 10.1007/978-3-319-46523-4_14.

[41] B. Ell, D. Vrandečić, and E. Simperl. SPARTIQULATION: Verbalizing SPARQL queries. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7540:117–131, 2015. ISSN 16113349. doi: 10.1007/978-3-662-46641-4_9.

[42] R. Farah, S. Hallé, J. Li, F. Lécué, B. Abeloos, D. Perron, J. Mattioli, P.-L. Gregoire, S. Laroche, M. Mercier, and P. Cocaud. Reasoning Engine for Support Maintenance. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12507 LNCS, pages 515–530. Springer International Publishing, 2020. ISBN 9783030624651. doi: 10.1007/978-3-030-62466-8_32.

[43] D. Fernàndez-Cañellas, J. Espadaler, D. Rodriguez, B. Garolera, G. Canet, A. Colom, J. M. Rimmek, X. Giro-i Nieto, E. Bou, and J. C. Riveiro. VLX-Stories: Building an Online Event Knowledge Base with Emerging Entity Detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11779 LNCS, pages 382–399. 2019. ISBN 9783030307950. doi: 10.1007/978-3-030-30796-7_24.

[44] S. Ferré. Expressive and Scalable Query-Based Faceted Search over SPARQL Endpoints. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, and C. Goble, editors, *The Semantic Web – ISWC 2014*, pages 438–453, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11915-1.

[45] S. Ferré. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web*, 8(3):405–418, 2017. ISSN 22104968. doi: 10.3233/SW-150208.

[46] A. Freitas and E. Curry. Natural language queries over heterogeneous linked data graphs: A distributional-compositional semantics approach. *International Conference on Intelligent User Interfaces, Proceedings IUI*, pages 279–288, 2014. doi: 10.1145/2557500.2557534.

[47] D. Garijo and M. Osorio. OBA: An Ontology-Based Framework for Creating REST APIs for Knowledge Graphs. In J. Z. Pan, V. Tamma, C. D'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, and L. Kagal, editors, *The Semantic Web – ISWC 2020*, pages 48–64, Cham, 2020. Springer International Publishing. ISBN 978-3-030-62466-8.

[48] Y. Gil, D. Garijo, V. Ratnakar, D. Khider, J. Emile-Geay, and N. McKay. A Controlled Crowdsourcing Approach for Practical Ontology Extensions and Metadata Annotations. In C. D'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, editors, *The Semantic Web – ISWC 2017*, pages 231–246, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68204-4.

[49] P. Grafkin, M. Mironov, M. Fellmann, B. Lantow, K. Sandkuhl, and A. V. Smirnov. SPARQL Query Builders : Overview and Comparison. In *BIR Workshops*, volume 1684 of *CEUR Workshop Proceedings*, pages 1–12. University of Rostock, Germany, CEUR-WS, 2016.

[50] J. C. Greene, V. J. Carcelli, and W. F. Graham. Toward a Conceptual Framework for Mixed-Method Evaluation Designs. *Educationl Evaluation and Policy Analysis*, 11(3):255–274, 1989.

[51] K. Gregory, P. Groth, H. Cousijn, A. Scharnhorst, and S. Wyatt. Searching Data: A Review of Observational Data Retrieval Practices in Selected Disciplines. *Journal of the Association for Information Science and Technology*, 70(5):419–432, may 2019. ISSN 2330-1635. doi: 10.1002/asi.24165.

[52] J. Groppe, S. Groppe, and A. Schleifer. Visual query system for analyzing social semantic web. *Proceedings of the 20th International Conference Companion on World Wide Web, WWW 2011*, pages 217–220, 2011. doi: 10.1145/1963192.1963293.

[53] F. Haag, S. Lohmann, and T. Ertl. Simplifying filter/flow graphs by subgraph substitution. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 145–148, 2012. ISSN 19436092. doi: 10.1109/VLHCC.2012.6344501.

[54] F. Haag, S. Lohmann, S. Bold, and T. Ertl. Visual SPARQL Querying Based on Extended Filter/Flow Graphs. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, AVI '14, pages 305–312, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327756. doi: 10.1145/2598153.2598185.

[55] F. Haag, S. Lohmann, and T. Ertl. SparqlFilterFlow: SPARQL Query Composition for Everyone. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, pages 362–367, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11955-7. doi: 10.1007/978-3-319-11955-7_49.

[56] F. Haag, S. Lohmann, S. Siek, and T. Ertl. QueryVOWL: Visual composition of SPARQL queries. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9341:62–66, 2015. ISSN 16113349. doi: 10.1007/978-3-319-25639-9_12.

[57] F. Haag, S. Lohmann, S. Siek, and T. Ertl. QueryVOWL: A Visual Query Notation for Linked Data. In *The Semantic Web: ESWC 2015 Satellite Events*, volume 9341, pages 387–402, 2015. ISBN 9783319256382. doi: 10.1007/978-3-319-25639-9_51.

[58] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. *Technical report, World Wide Web Consortium (W3C)*, 2013. URL `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[59] S. G. Hart and L. E. Staveland. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In P. A. Hancock and N. Meshkati, editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. North-Holland, 1988. doi: 10.1016/S0166-4115(08)62386-9.

[60] S. Heggestøyl, G. Vega-Gorgojo, and M. Giese. Visual Query Formulation for Linked Open Data: The Norwegian Entity Registry Case Simen Heggestøyl. *Norsk Informatikkonferanse (NIK)*, 2014.

[61] P. Heim and J. Ziegler. Faceted Visual Exploration of Semantic Data. In A. Ebert, A. Dix, N. D. Gershon, and M. Pohl, editors, *Human Aspects of Visualization*, pages 58–75, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19641-6.

[62] P. Hoefler, M. Granitzer, E. Veas, and C. Seifert. Linked data query wizard: A novel interface for accessing sparql endpoints. *CEUR Workshop Proceedings*, 1184(January), 2014. ISSN 16130073.

[63] K. Höffner and J. Lehmann. Towards Question Answering on Statistical Linked Data. In *Proceedings of the 10th International Conference on Semantic Systems*, SEM '14, pages 61–64, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329279. doi: 10.1145/2660517.2660521.

[64] K. Höffner, J. Lehmann, and R. Usbeck. CubeQA—Question Answering on RDF Data Cubes. In *The Semantic Web – ISWC 2016*, volume 9981 of *Lecture Notes in Computer Science*, pages 325–340, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46522-7. doi: 10.1007/978-3-319-46523-4_20.

[65] K. Höffner, S. Walter, E. Marx, R. Usbeck, J. Lehmann, and A. C. Ngonga Ngomo. Survey on challenges of Question Answering in the Semantic Web. *Semantic Web*, 8(6):895–920, 2017. ISSN 22104968. doi: 10.3233/SW-160247.

[66] F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: A SPARQL-based graphical query language for RDF. *Advanced Information and Knowledge Processing*, 53:87–116, 2010. ISSN 21978441. doi: 10.1007/978-1-84996-074-8_4.

[67] S. Holmås, R. R. Puig, M. L. Acencio, V. Mironov, and M. Kuiper. The cytoscape BioGateway app: Explorative network building from an RDF store. *Bioinformatics*, 36(6):1966–1967, 2020. ISSN 14602059. doi: 10.1093/bioinformatics/btz835.

[68] D. Huynh and D. Karger. Parallax and Companion: Set-based Browsing for the Data Web. In *IW3C2*, 2009. ISBN 9781595936547.

[69] K. Janowicz, A. Haller, S. J. D. Cox, D. Le Phuoc, and M. Lefrançois. SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics*, 56:1–10, 2019. ISSN 1570-8268. doi: https://doi.org/10.1016/j.websem.2018.06.003. URL `https://www.sciencedirect.com/science/article/pii/S1570826818300295`.

[70] A. Jares and J. Klimek. *Simplod: Simple SPARQL Query Builder for Rapid Export of Linked Open Data in the Form of CSV Files*, pages 415–418. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450395564.

[71] E. Kaufmann and A. Bernstein. How useful are natural language interfaces to the semantic Web for casual end-users? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4825 LNCS:281–294, 2007. ISSN 03029743. doi: 10.1007/978-3-540-76298-0_21.

[72] E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. *5th International Semantic Web Conference (ISWC 2006)*, (November):980–981, 2006.

[73] E. Kaufmann, A. Bernstein, and L. Fischer. NLP-Reduce: A "naïve" but Domain-independent Natural Language Interface for Querying Ontologies. *4th European Semantic Web Conference (ESWC 2007)*, pages 1–2, 2007.

[74] M. Kejriwal and P. Szekely. An Investigative Search Engine for the Human Trafficking Domain. In C. D'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, editors, *The Semantic Web – ISWC 2017*, volume 10588 LNCS, pages 247–262, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68204-4. doi: 10.1007/978-3-319-68204-4_25.

[75] B. Kitchenham and P. Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013. ISSN 0950-5849. doi: 10.1016/j.infsof.2013.07.010.

[76] J. Klímek, P. Škoda, and M. Nečaský. Survey of tools for Linked Data consumption. *Semantic Web*, 10:665–720, 2019. ISSN 2210-4968. doi: 10.3233/SW-180316.

[77] G. Kobilarov and I. Dickinson. Humboldt: Exploring linked data. In *CEUR Workshop Proceedings*, volume 369, 2008.

[78] E. Kuric, J. D. Fernández, and O. Drozd. Knowledge Graph Exploration: A Usability Evaluation of Query Builders for Laypeople. In *Semantic Systems. The Power of AI and Knowledge Graphs*, volume 11702 LNCS, pages 326–342, Cham, 2019. Springer International Publishing. ISBN 9783030332198. doi: 10.1007/978-3-030-33220-4_24.

[79] N. Lasolle, O. Bruneau, J. Lieber, E. Nauer, and S. Pavlova. Assisting the RDF Annotation of a Digital Humanities Corpus Using Case-Based Reasoning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12507 LNCS, pages 617–633. Springer International Publishing, 2020. ISBN 9783030624651. doi: 10.1007/978-3-030-62466-8_38.

[80] D. Lembo, D. Pantaleone, V. Santarelli, and D. F. Savo. Easy OWL Drawing with the Graphol Visual Ontology Language. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'16, pages 573–576. AAAI Press, 2016.

[81] M. Lenzerini. Ontology-Based Data Management. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 5–6, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307178. doi: 10.1145/2063576.2063582.

[82] P. Leskinen, M. Koho, E. Heino, M. Tamper, E. Ikkala, J. Tuominen, E. Mäkelä, and E. Hyvönen. Modeling and Using an Actor Ontology of Second World War Military Units and Personnel. In C. D'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, editors, *The Semantic Web – ISWC 2017*, pages 280–296, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68204-4.

[83] P. Lisena, A. Meroño-Peñuela, T. Kuhn, and R. Troncy. Easy Web API Development with SPARQL Transformer. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11779 LNCS, pages 454–470. Springer International Publishing, 2019. ISBN 9783030307950. doi: 10.1007/978-3-030-30796-7_28.

[84] S. Lohmann, V. Link, E. Marbach, and S. Negru. WebVOWL: Web-based Visualization of Ontologies. In P. Lambrix, E. Hyvönen, E. Blomqvist, V. Presutti, G. Qi, U. Sattler, Y. Ding, and C. Ghidini, editors, *Knowledge Engineering and Knowledge Management*, pages 154–158, Cham, 2015. Springer International Publishing. ISBN 978-3-319-17966-7.

[85] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016. ISSN 22104968. doi: 10.3233/SW-150200.

[86] V. Lopez, S. Kotoulas, M. L. Sbodio, and R. Lloyd. Guided Exploration and Integration of Urban Data. In *Proceedings of the 24th ACM Conference on Hypertext and Social Media*, HT '13, pages 242–247, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319676. doi: 10.1145/2481492.2481524.

[87] V. Lopz, M. Stephenson, S. Kotoulas, and P. Tommasi. Finding mr and mrs entity in the city of knowledge. *HT 2014 - Proceedings of the 25th ACM Conference on Hypertext and Social Media*, pages 261–266, 2014. doi: 10.1145/2631775.2631817.

[88] S. Malyshev, M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *17th International Semantic Web Conference*, volume 11137 LNCS, pages 376–394, 2018. ISBN 9783030006679. doi: 10.1007/978-3-030-00668-6_23.

[89] L. McCarthy, B. Vandervalk, and M. Wilkinson. SPARQL assist language-neutral query composer. *BMC bioinformatics*, 13 Suppl 1(Suppl 1):S2, 2012. ISSN 14712105. doi: 10.1186/1471-2105-13-s1-s2.

[90] A. Meroño-Peñuela and R. Hoekstra. Automatic query-centric API for routine access to linked data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10588 LNCS, pages 334–349, 2017. ISBN 9783319682037. doi: 10.1007/978-3-319-68204-4_30.

[91] N. Mihindukulasooriya, R. Mahindru, M. F. M. Chowdhury, Y. Deng, N. R. Fauceglia, G. Rossiello, S. Dash, A. Gliozzo, and S. Tao. Dynamic Faceted Search for Technical Support Exploiting Induced Knowledge. pages 683–699. Springer International Publishing, 2020. ISBN 9783030624668. doi: 10.1007/978-3-030-62466-8_42.

[92] D. S. Mishra, A. Agarwal, B. P. Swathi, and K. C. Akshay. Natural language query formalization to SPARQL for querying knowledge bases using Rasa. *Progress in Artificial Intelligence*, 2021. ISSN 2192-6360. doi: 10.1007/s13748-021-00271-1.

[93] R. Mulero, V. Urosevic, A. Almeida, and C. Tatsiopoulos. Towards ambient assisted cities using linked data and data analysis. *Journal of Ambient Intelligence and Humanized Computing*, 9(5):1573–1591, 2018. ISSN 1868-5145. doi: 10.1007/s12652-018-0916-y.

[94] A.-C. N. Ngomo, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber. SPARQL2NL-Verbalizing SPARQL queries. In *Proceedings of the 22nd International Conference on World Wide Web - WWW '13 Companion*, New York, New York, USA, 2013. ACM Press. ISBN 9781450320382.

[95] A. C. Ngonga, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber. Sorry, i don't speak SPARQL - Translating SPARQL queries into natural language. *WWW 2013 - Proceedings of the 22nd International Conference on World Wide Web*, pages 977–987, 2013.

[96] J. Nielsen and T. K. Landauer. A Mathematical Model of the Finding of Usability Problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 206–213, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897915755. doi: 10.1145/169059.169166.

[97] F. Osborne, A. Salatino, A. Birukou, and E. Motta. Automatic classification of springer nature proceedings with smart topic miner. In *The Semantic Web - ISWC 2016*, pages 383–399, 2016. ISBN 9783319465463. doi: 10.1007/978-3-319-46547-0_33.

[98] T. Pankowski. Ontological databases with faceted queries. *The VLDB Journal*, 2022. ISSN 0949-877X. doi: 10.1007/s00778-022-00735-3.

[99] G. M. R. I. Rasiq, A. A. Sefat, T. Hossain, M. I.-E.-H. Munna, J. J. Jisha, and M. M. Hoque. Question Answering System over Linked Data: A Detailed Survey. *ABC Research Alert*, 8(1):32–47, 2020. doi: 10.18034/abcra.v8i1.449.

[100] L. Rietveld and R. Hoekstra. The YASGUI family of SPARQL clients 1. *Semantic Web*, 8(3):373–383, 2017. ISSN 22104968. doi: 10.3233/SW-150197.

[101] D. Rough and A. Quigley. Challenges of Traditional Usability Evaluation in End-User Development. In A. Malizia, S. Valtolina, A. Morch, A. Serrano, and A. Stratton, editors, *End-User Development*, pages 1–17, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24781-2.

[102] A. Russell and P. R. Smart. NITELIGHT: A graphical editor for SPARQL queries. *CEUR Workshop Proceedings*, 401:2–3, 2008. ISSN 16130073.

[103] A. Russell, P. R. Smart, D. Braines, and N. R. Shadbolt. NITELIGHT: A graphical tool for semantic query construction. *CEUR Workshop Proceedings*, 543(November), 2009. ISSN 16130073.

[104] A. A. Salatino, T. Thanapalasingam, A. Mannocci, F. Osborne, and E. Motta. The Computer Science Ontology: A Large-Scale Taxonomy of Research Areas. In D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, editors, *The Semantic Web - ISWC 2018*, pages 187–205, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00668-6. doi: 10.1007/978-3-030-00668-6_12.

[105] A. A. Salatino, F. Osborne, A. Birukou, and E. Motta. Improving Editorial Workflow and Metadata Quality at Springer Nature. In *The Semantic Web - ISWC 2019*, pages 507–525. Springer International Publishing, 2019. ISBN 9783030307950. doi: 10.1007/978-3-030-30796-7_31.

[106] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A.-C. N. Ngomo. LSQ: The Linked SPARQL Queries Dataset. In *The Semantic Web - ISWC 2015*, volume 9367, pages 261–269, 2015. ISBN 9783319250090. doi: 10.1007/978-3-319-25010-6.

[107] W. Sebastian, U. Christina, C. Philipp, and B. Daniel. Evaluation of a Layered Approach to Question Answering over Linked Data. *The Semantic Web – ISWC 2012*, 7650(00):362–374, 2012. doi: 10.1007/978-3-642-35173-0.

[108] S. Shekarpour and S. Auer. SINA: semantic interpretation of user queries for question answering on interlinked data. *ACM SIGWEB Newsletter*, (Summer):1–1, 2014. ISSN 1931-1745. doi: 10.1145/2641730.2641733.

[109] P. R. Smart, A. Russell, D. Braines, Y. Kalfoglou, J. Bao, and N. R. Shadbolt. A visual approach to semantic query design using a web-based graphical query designer. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5268 LNAI:275–291, 2008. ISSN 16113349. doi: 10.1007/978-3-540-87696-0_25.

[110] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie, and I. Horrocks. OptiqueVQS: A visual query system over ontologies for industry. *Semantic Web*, 9(5):627–660, 2018. ISSN 15700844. doi: 10.3233/sw-180293.

[111] T. Thanapalasingam, F. Osborne, A. Birukou, and E. Motta. Ontology-based recommendation of editorial products. In *The Semantic Web - ISWC 2018*, pages 341–358, 2018. ISBN 9783030006679. doi: 10.1007/978-3-030-00668-6_21.

[112] C. Unger, A. Freitas, and P. Cimiano. An introduction to question answering over linked data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8714:100–140, 2014. ISSN 16113349. doi: 10.1007/978-3-319-10587-1_2.

[113] H. Vargas, C. Buil-Aranda, A. Hogan, and C. López. RDF Explorer: A Visual SPARQL Query Builder. In *The Semantic Web – ISWC 2019*, volume 11778 LNCS, pages 647–663, Cham, 2019. Springer International Publishing. ISBN 9783030307929. doi: 10.1007/978-3-030-30793-6_37.

[114] G. Vega-Gorgojo, L. Slaughter, M. Giese, S. Heggestøyl, A. Soylu, and A. Waaler. Visual query interfaces for semantic datasets: An evaluation study. *Journal of Web Semantics*, 39:81–96, 2016. ISSN 1570-8268. doi: 10.1016/j.websem.2016.01.002.

[115] P. Warren and P. Mulholland. A Comparison of the Cognitive Difficulties Posed by SPARQL Query Constructs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12387 LNAI, pages 3–19. Springer International Publishing, 2020. ISBN 9783030612436. doi: 10.1007/978-3-030-61244-3_1.

[116] E. Wittern, A. T. T. Ying, Y. Zheng, J. Dolby, and J. A. Laredo. Statically Checking Web API Requests in JavaScript. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 244–254, 2017. doi: 10.1109/ICSE.2017.30.

[117] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries—Incremental query construction on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7 (3):166–176, 2009. ISSN 15708268. doi: 10.1016/j.websem.2009.07.005.

[118] W. Zheng, H. Cheng, L. Zou, J. X. Yu, and K. Zhao. Natural language question/answering: Let users talk with the knowledge graph. *International Conference on Information and Knowledge Management, Proceedings*, Part F1318:217–226, 2017. doi: 10.1145/3132847.3132977.

[119] M. Zviedris and G. Barzdins. ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, and J. Pan, editors, *The Semanic Web: Research and Applications*, pages 441–445, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21064-8. doi: 10.1007/978-3-642-21064-8_31.

[120] U. Şimşek, K. Angele, E. Kärle, O. Panasiuk, and D. Fensel. Domain-Specific Customization of Schema.org Based on SHACL. volume 12507 of *Lecture Notes in Computer Science*, pages 585–600, Cham, 2020. Springer International Publishing. ISBN 978-3-030-62466-8. doi: 10.1007/978-3-030-62466-8_36.