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 23, 2022

ABSTRACT

Linked-Data (LD) is a paradigm that utilises the RDF triplestore to describe numerous pieces of knowledge linked together. When an entity is retrieved in LD, its associated data becomes instantly obtainable. SPARQL is the query language that allows users to access LD. On the other hand, SPARQL has a complicated syntax that necessitates previous knowledge. Thus, in order to encourage the end-users to use LD, it is crucial to allow them to obtain the data efficiently, in addition to improving their overall experience. Instead of manually constructing SPARQL queries, this paper investigates and reviews existing methods in which LD can be accessed using various tools and techniques, including query builders, visualisation approaches, and several LD applications. We then identify gaps within the literature and highlight future research directions.

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

1 Introduction

Since the introduction of Linked-Data (LD), competition has arisen between various organisations to publish data in a widely adoptable machine-readable format, typically as Resource Description Framework (RDF)[34, 25]. An RDF consists of triples that are structured as interlinked entities to make them both accessible and machine-readable. The volume of published data has thus rapidly expanded to incorporate billions of triples, greatly increasing the difficulty of navigating such data to extract useful information [34, 25, 63].

LD is also known as Linked Open Data (LOD) if data used comes from open sources. Many LOD are available through SPARQL endpoints, which thus receive millions of SPARQL queries per day, both from humans and machine [20, 105, 88]. Although LD makes data machine-readable, humans still constitute a considerable proportion of LD consumers [88, 21]. SPARQL is the W3C recommended Query Language for accessing LD [59]. SPARQL is a powerful and precise query language. This ensures that even though LD contains billions of triples and relationships, the user can specify the exact relationship that matches their explicit patterns and constraints using SPARQL. However, SPARQL is complex and creates a high cognitive load, and learning SPARQL can be very challenging, especially initially [114, 63, 7]. Thus, mainstream users cannot access LD using SPARQL without direct assistance, though experts find it less intimidating. Nevertheless, writing accurate SPARQL queries requires an understanding of the underlying ontology, which reflects the scheme that defines the relationships and terms between distinct entities. This includes names of the namespaces, vocabularies, and data, yet many SPARQL endpoints lack a "Unique name assumption" and are not adequately documented; they may even not be documented at all [118, 29, 18]. As a result, developers may try to guess at the appropriate ontology and hope to hit upon the correct names. Thus, even SPARQL experts may struggle in creating SPARQL queries to retrieve data from an unfamiliar SPARQL endpoint [63].

Based on the complexity of SPARQL, why is it being used? The complexity of SPARQL arises mainly from the complexity of the underlying data structure and the intended queries, as the language itself allows high expressivity, and SPARQL is not the only language that can be used to query knowledge graphs. Languages such as Cypher or Gremlin are also used for such queries; however, these languages are also not suitable for the Lay-users [5, 98, 59]. In addition, in terms of querying RDF triplestores, SPARQL is the leading standard as endorsed by the W3C, and it also excels in querying heterogeneous and distributed datasets that are semantically linked [5, 59].

The development of a user-friendly interface to assist users in creating SPARQL queries and exploring data effectively is thus urgent. There have been numerous efforts to develop a SPARQL Query Builder (QB) Interface. These SPARQL QBs are End-User Development (EUD) tools that aim to enable users to construct SPARQL queries freely by manipulating graphical interface components [9, 15]. EUD tools are designed to overcome the difficulty by providing an alternative that is crafted to fulfil the end-user requirements and skills [101]. However, many of these interfaces still have steep learning curves and require an in-depth understanding of the ideas behind LD and the ways in which data is connected [112, 78]. Some of these tools were created to enhance experiences within the Semantic Web community, while other tools are focused on supporting experts and are thus domain-dependant. In contrast, many mainstream or lay users have expressed a preference for tools to query LD that do not require previous experience with SPARQL or Semantic Web. For the purposes of this study, users are divided into three main types:

- **Lay-users**: These users have no previous knowledge or understanding of SPARQL or Semantic Web.
- **Tech-users**: These users understand the Web Semantic domain and how LD works, and they may even be able to read SPARQL and understand its meaning; however, they have not been directly involved previously in manually creating SPARQL queries.
- **Expert-users**: These are familiar with SPARQL and can develop complex SPARQL queries. They have previously worked with RDF and other Semantic Web technologies.

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 rest of this paper is organised as follows: Section 2 explains the search methodology used, while Section 3 the explores tools and approaches available that offer access to LD, as well as introducing SPARQL Query Builders. Section 4 then analyses and discusses some of the findings, allowing Section 5 to focus on the emergent research challenges and limitations. Section 5 then concludes the work by offering some final thoughts.

2 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 1 summarises the search stages as well as the selection technique.

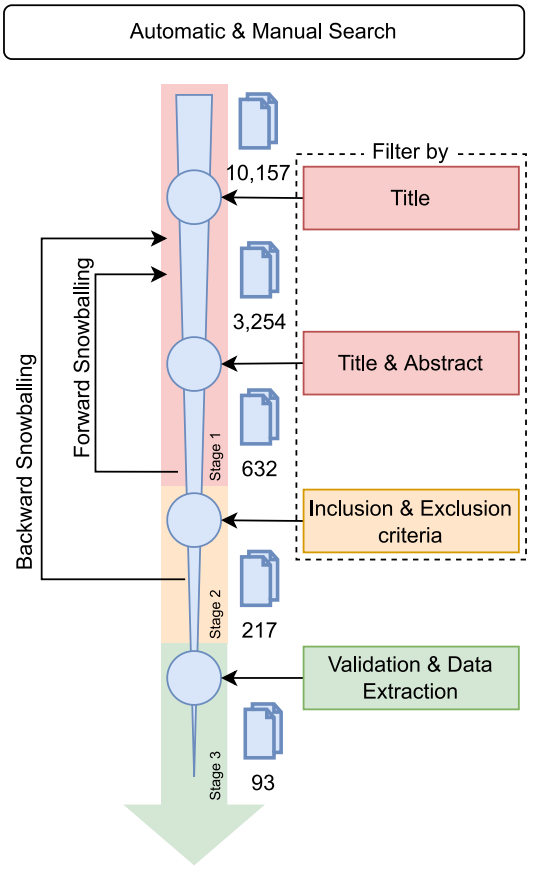

Figure 1: Demonstration of the search stages with the selection strategy.

2.1 Search Strategy

Influenced by Kitchenham and Brereton [76] 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 journals or conference proceedings verified by experts such as "Semantic Web journal", "International Working Conference on Advanced Visual Interfaces", and "International Conference on Semantic Systems". For section 4, we have reviewed 33 out of 145 paper that was published as in-use or resource during 2021 - 2015 in the International Semantic Web Conference.

2.2 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.

2.2.1 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.

2.3 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.

3 Accessing Linked Data

This section, explores the tools available to obtain data from SPARQL endpoints. Tool categorisation was based on the visual approach of the data retrieval process (see Figure 2). The Visual Query Builders classification was inspired by Grafkin et al. [50], Kuric et al. [78], Vargas et al. [112] to represent the visual interface based on its querying approach. The other UIs' categorisation was suggested by various different papers, which are mentioned separately in each relevant section.

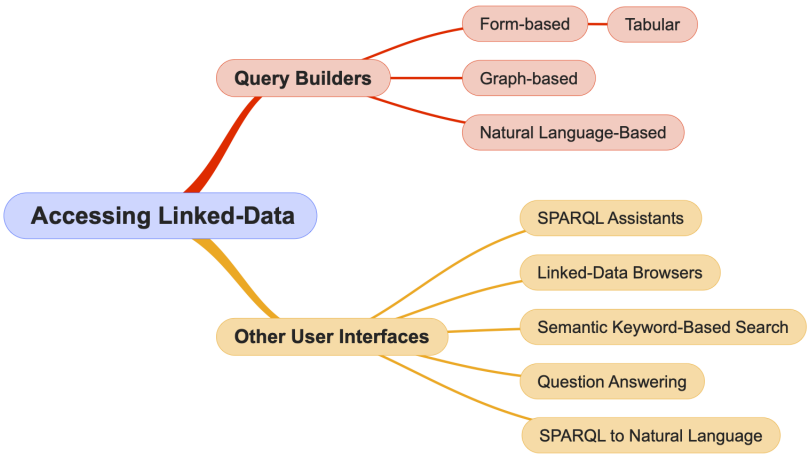

Figure 2: A diagram that illustrates the reviewed tools' classification for obtaining data from SPARQL endpoints.

3.1 Query Builders

3.1.1 Form-based

The Konduit VQB [4] interface introduced an approach to formulate "CONSTRUCT", or "SELECT" queries based on "schema" or "instance". It is a Form-Based interface designed for Expert-users and Tech-users that facilitates the creation of SPARQL queries. The "Schema-based" interface takes advantage of the SPARQL's tree-like structure to simplify the underlying interface component, using a top-down structure with field indentation to reflect the parent-child relationship. In contrast, the "Instance-based" interface adopts the RDF triple (Subject, Predicate and Object) structure, allowing for more precise querying with less abstraction, which may be excessively obscure for Lay-users. In spite of that, this interface was not evaluated.

The BioGateway Query Builder App [68] is a desktop application designed to query the BioGateway SPARQL endpoint. The BioGateway App is similar to the Konduit VQB Schema-based interface in that it uses multiple drop-down lists in a Form-based interface. However, the BioGateway App does not utilise the child-parent structure seen in Konduit VQB; instead, each line represents an RDF triple that can be linked to the ensuing line using predefined variables (Set A, Set B, etc.). A significant feature of the BioGateway App that is not found in Konduit VQB is its predefined examples. The BioGateway App includes a set of predefined examples that promote learning by example. Despite that, the app is more domain-specific and requires some understanding of the BioGateway ontology and SPARQL, and even Tech-users are thus likely to face challenges using this tool.

Wikidata Query Service (WQS) [1] is the Wikidata official query tool to run SPARQL queries. The tool offers a query builder known as "Query Helper" to assist users in creating or modifying SPARQL queries where they lack prior familiarity with SPARQL [88] (see Figure 3). The Query Helper is a Form-Based interface that relies mainly on drop-down lists. The idea is based on the "Filters" and "Show" interface. The interface will help the users to find an item that must match the listed filters. In addition, users can add some optional matches using the "Show" section to be presented when the data is retrieved. The right-hand side of the interface also displays the auto-generated SPARQL query, which allows Tech-users and Expert-users to manually modify the SPARQL query and get it reflected on the Query Helper. For Lay-users, however, this feature is not desirable and might be seen as overwhelming for some users [78].

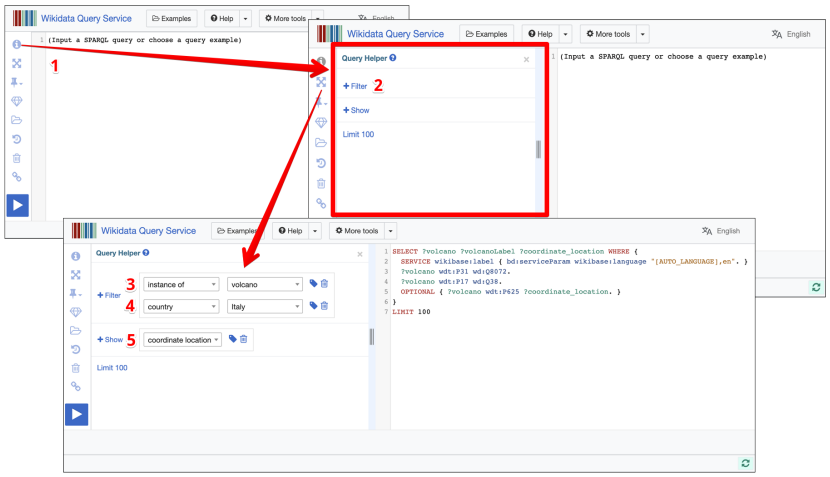

Figure 3: An example of using the Query Helper interface in Wikidata Query Service to generate SPARQL to extract all of the "Italian Volcanoes with its coordinates if exist". (1) Clicking on the button to show Query Helper. (2) Clicking on Filter. (3) Writing "Volcano" as the wanted entity and select "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.

VizQuery [2] is an elementary query builder that relies on matching the rules, which are RDF triple patterns. The interface assists the user in finding those items that match the selected rules. For example, if a user wants to find all "Cats" in the dataset, they must select the property "instance of" and the item "house cat" (see Figure 4). Unlike WQS, this

[1] `https://query.wikidata.org/`

[2] `https://hay.toolforge.org/vizquery/`

tool offers no way to add some optional patterns or create a more complex structure. However, as with WQS, the tool supports auto-complete and provides a predefined example to enhance the user experience.

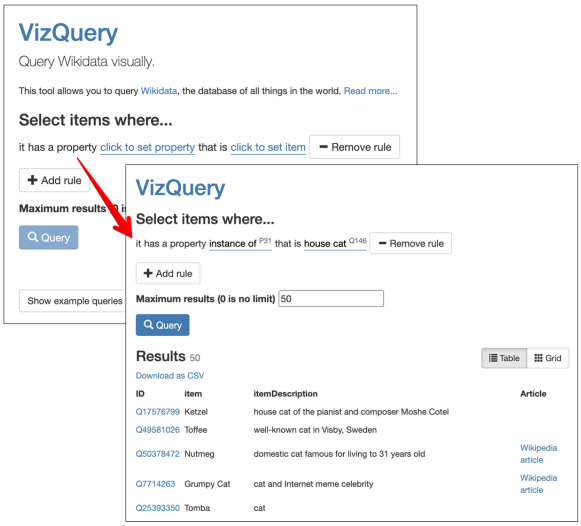

Figure 4: An example of using the VizQuery interface to find all indoor Cats in Wikidata dataset by selecting the property "instance of" and the item "house cat".

One of the highly interactive query builders is SparqlFilterFlow [56], which is based on the idea of FILTER/FLOW MODEL that was introduced in 1993 as an SQL query builder. The interface is Form-based with graph representation, making it more intuitive and allowing users to understand the relationships between the entities. The user adds some entity as a starting point, then creates some filters to narrow down the results [54]. Coloured links occur between all of the interface components to support the top-down flow of data, and the links between the graph entities reflect the amount of data present by varying in thickness. The author conducted a user study to examine the interface, and all participants were able to solve all of the tasks. The most interesting thing about this study was that participants could also comprehend and fix their mistakes to achieve the correct output. One of the great features that helped the participants adjust their input was the path thickness, which helped them understand the amount of data they were getting and to respond accordingly. The original prototype presented by Haag et al. [55] was a desktop application, but the authors have created a web-based demo [3] for the same approach with limited features (see Figure 5).

Simplod [71] is a tool that aims to assist the user in extracting data in bulk from LOD as a CSV file. The tool relies on the LOD schema being loaded as RDF Turtle into the tool in order to construct the query. The Dataset schema view is represented on the left side of the interface, and the entities are given in the form of a UML Class diagram to describe the relationships. The schema list view, on the right side of the interface, displays the schema as a list with checkboxes indicating whether these are included in the search or not. Simplod can only generate SELECT SPARQL queries, and it has no customisation or filtration capabilities. As a result of this limitation, Simplod is only useful for investigating the LOD and downloading the results in bulk. During their evaluation, the SUS score for this tool was 63.125 for users with no experience with RDF, which is below the average of 68. However, experienced users have scored 72.5, suggesting that this tool requires training and is thus most suitable for Tech-users.

Konduit VQB, BioGateway App, WQS, VizQuery and SparqlFilterFlow are all similar in that they rely on the visualisation of the RDF triple structure based on the use of prefilled drop lists. The user must thus select an item and add some results filtration. While Konduit VQB and BioGateway App require some SPARQL knowledge, WQS, VizQuery, and SparqlFilterFlow present more straightforward designs. VizQuery can also be considered to be a simpler version of WQS design as VizQuery rules follows the same filter scheme (see Table 1).

Tabular Query Builders To take advantage of user familiarity with spreadsheets, Tabular Query Builders will usually present the LD in abular format that allows the user to add some filters and constraints to narrow down the results in a similar manner to spreadsheet software. Examples of such query builders include Linked Data Query Wizard (LDQW) [63], and ExConQuer [7].

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

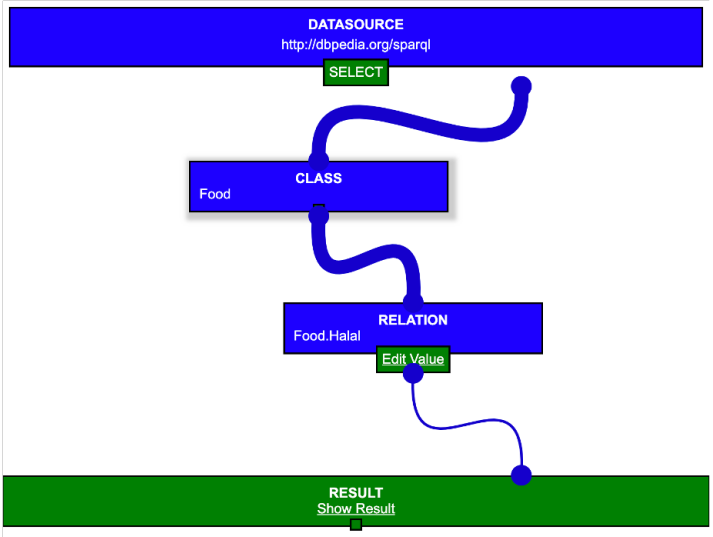

Figure 5: An example of using SparqlFilterFlow's online demo to get all of the entities that is Food and also Halal. The thickness of the lines reflect the results size.

Table 1: Reviewed Form-based SPARQL Query Builders

Tool	User Type [a]	Environment [b]	Is Evaluated?	SPARQL [c]
Konduit VQB [4]	E,T	Desktop	✗	✓
SPARQLViz [26]	E,T	Desktop	✗	✓
PepeSearch [61]	T,L	Web	✓	✗
GPB [12]	E,T	Web	✗	✗
BioGateway App [68]	E	Desktop	✗	✓
WQS [78]	E,T	Web	✓	✓
VizQuery	All	Web	✗	✗
SparqlFilterFlow [56]	All	Desktop/Web	✓	✗
Simplod [71]	T	Web	✓	✓
LDQW [63]	All	Web	✓	✓
ExConQuer [7]	E,T	Web	✗	✓
Falcons Explorer [31]	E,T	Web	✗	✗

[a] E: Expert-users, T: Tech-users, L: Lay-users, All: All users.
[b] Desktop: Desktop-App, Web: Web-based App.
[c] Does the tool allow the user to view SPARQL query?

Linked Data Query Wizard (LDQW) is a SPARQL query builder that utilises user familiarity with search engines and spreadsheet software to flatten its learning curve and increase the user abstraction from the LD concept. The LDQW asks the user to start with a keyword, as with any search engine, then takes the user to another page showing some initial results in a tabular format. The user can then begin digging into the data by applying filters based on the available properties. The user can add new columns into the table to represent some of the available hidden data. Additionally, the user can represent the data in a mind map format or visualise it using predefined charts. LDQW also grants users the option to review the generated SPARQL queries with some statistics about the run time, which is an outstanding feature for Experts and Tech-users. In general, the conducted usability study results were encouraging, though they did indicate several design vulnerabilities that can be improved.

ExConQuer is another SPARQL query builder that uses the tabular format to help the user to become familiar with the interface. However, rather than displaying a table with some initial results as in LDQW, ExConQuer displays the selected subject's properties and allows the user to choose from these properties to add some data filters and narrow down the results. ExConQuer thus relies on the Faceted-Browsing concept as a way to not overwhelm the users, splitting the interface into three main steps. The initial step involves selecting the data source; the user is then asked to pick a single subject which is called a concept. Finally, users will get all the data about that subject, although they may pick further properties to be matched when querying the data. Before running the query, a text box that contains the auto-generated SPARQL appears, making it possible to modify the SPARQL before running it manually. According to the authors, the purpose of this feature is to help users learn about and familiarise themselves with SPARQL and RDF, in addition to enhancing the experts' use experience. The tool was examined by Experts and Tech-users in comparison with manually creating SPARQL queries. These users were asked to perform the same task with and without the tool and to rate their experiences in both scenarios. The results showed that the tools were beneficial for all users, especially those with less experience with SPARQL.

Falcons Explorer [31] is a LD Browser built on top of "Falcons Object Search" with a built-in query builder. This tool is a less clear fit for this section, as it resembles a regular LD browser that supports faceted browsing. However, the tool has some similarities with LDQW. Users must begin by typing an entity name into the search box. The results can be represented in a tabular format, where the users can pick a specific entity that will take them to the entity page with all the properties and information linked with that entity. The table of results is similar to a spreadsheet, supporting several methods of filtering results and managing the table columns. In addition to the tabular format, the tool supports presenting data in a relational format similar to a relational database design. This relational design allows the user to use some typical relational operations such as Projection and Joins. The interface is very complicated due to the existence of multiple ways to access the data and retrieve the desired results, making it potentially confusing for Lay-users.

3.1.2 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 6). 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.

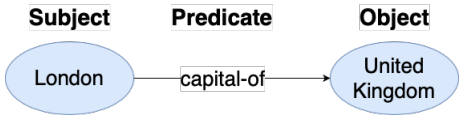

Figure 6: An example of RDF basic triple pattern.

One of the earliest attempts to create SPARQL Query Builder is the Graphical Query Language (GQL) Tool [18]. The authors have followed "Ontology Definition Metamodel"[4] 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.

Rather than representing RDF in UML class diagrams as seen in the GQL Tool, Hogenboom et al. [67] 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.

SPARQLING [17] 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 [33]. 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

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

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.

One of the most well-known query builders is the OpenLink interactive SPARQL Query Builder (iSPARQL) [5]. 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 [108].

NITELIGHT[102] 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 offering a more realistic and attractive graphical representation [108]. 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.

ViziQuer [118] 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 7 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 [6].

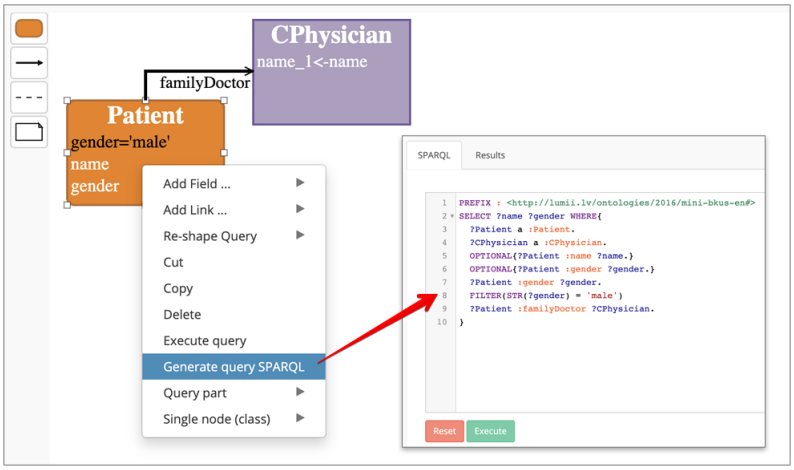

Figure 7: An example of using ViziQuer to generate SPARQL query that will extract the names of all male patients who have a family doctor.

QueryVOWL [57] 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)[57]. 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

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

[6] https://viziquer.lumii.lv

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].

OptiqueVQS [109] 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 [109]. 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 [109, 78].

RDF Explorer [112] 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[112], 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 [55].

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.

3.1.3 Natural Language-Based

For Lay-users, nothing is more comfortable than using the natural language. Multiple tools have thus tried to use natural language to generate SPARQL. Most of these attempts have focused on creating a questions answering system that will output a YES/NO or a single statement response in each case. This section, however, presents only those tools that will either generate SPARQL queries or allow the user to retrieve multiple results.

Querix [73] 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 [72].

NLP-Reduce [74] 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

Table 2: Reviewed Graph-based SPARQL Query Builders

Tool	Metaphor [a]	User Type [b]	Environment	Is Evaluated?	SPARQL [d]
GQL tool [18]	UML	E,T	Desktop	✗	✗
SPARQLinG [67]	RDF-GL	E,T	Desktop	✗	✓
SPARQLING [17]	GRAPHOL	E	Web	✗	✓
iSPARQL	RDF-Triple	E	Web	✗	✓
NITELIGHT [102]	RDF-Triple	E,T	Web	✗	✓
ViziQuer [118]	RDF-Triple	E,T	Web	✗	✓
VQS [53]	RDF-Triple	E,T	Web	✗	✓
QueryVOWL [57]	VOWL	All	Web	✓	✓
RDF Explorer [112]	RDF-Triple	All	Web	✓	✓
OptiqueVQS [109]	RDF-Triple	All*	Web	✓	✓

[a] The used graphical metaphor for the user interface.
[b] E: Experts, T: Tech-users, L: Lay-users, All: All users.
[d] Does the tool allow the user to view SPARQL query?
* Only users with industrial background.

three distinct rounds to generate SPARQL queries. Ultimately, the highest-scoring query will be displayed to the user. Thus, users can also use non-complete sentences or even keywords [74]. NLP-Reduce has been proven to act as a rapid query-building tool, particularly in comparison to certain similar tools [72]. 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.

While most of the keywords NL-based query builders lack the expressivity to capture the user intention, QUICK [116] 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 [116].

SPARKLIS [46] 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].

onIQ [39] is a tool that translates queries in natural language into SPARQL. Unlike Querix and QUICK, onIQ does not use Stanford Parser. Instead, it uses spaCy library [7] which the author assumes to gain a better performance over the other solutions. onIQ was not evaluated, so this assumption may not be valid. 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.

[7] https://spacy.io/

Table 3: Reviewed Natural Language-Based SPARQL Query Builders

Tool	Search Type [a]	User Type [b]	Environment [c]	Is Evaluated?	SPARQL [d]
Querix [73]	Keyword	T, L	Desktop	✓	✓
NLP-Reduce [74]	Keyword	T, L	Desktop	✓	✓
QUICK [116]	Keyword	All	Web	✓	✗
SPARKLIS [46]	Sentence	All	Web	✓	✓
onIQ [39]	Keyword	T	Web	✗	✓

[a] The tool search scheme.
[b] E: Experts, T: Tech-users, L: Lay-users, All: All users.
[c] Desktop: Desktop App, Web: Web-based App.
[d] Does the tool allow the user to view SPARQL query?

3.2 Other User Interfaces

This section explores several tools that will facilitate access to LD that differ from SPARQL query builders by not focusing directly on constructing the queries. Instead, it focuses on browsing the data and finding an exact answer. The section will include SPARQL Assistants, LD Browsers and Question Answering Systems. Then, it concluded with a discussion about SPARQL to Natural Language tools.

3.2.1 SPARQL Assistants

Assuming the user has experience with SPARQL, query editing interfaces provide more flexibility and enable precise querying. These tools can help experts become familiar with the SPARQL endpoint by offering suggestions and autocomplete features. Examples of such tools include SPARQL Assist [89], YASGUI [100], and WQS [88].

SPARQL Assist is a web-based SPARQL query editor intended for less experienced SPARQL endpoint consumers. The tool assists users by showing all possible options that match the user's cursor position in a query. For example, if the user type 'Ca' as a 'Subject', it will display a list of subjects that start with 'Ca'. The list will also include a description for each item where possible. Thus, constructing a manual SPARQL query becomes a guided process that does not require any previous knowledge about the ontology. Furthermore, the tool also supports multi-language terms by examining resource labels to find any possible matches.

YASGUI and WQS are similar to SPARQL Assist, with fewer restrictions and fewer guiding features, making them more suitable for experienced users. YASGUI is a general-purpose SPARQL editor created to be used as an endpoint editor, while WQS acts as an editor only for Wikidata. Unlike SPARQL Assist, neither tool supports multi-language terms. However, as discussed in the previous section, WQS offers a form-based SPARQL query builder that is tightly coupled with the text editor, reflecting most of the user input. In contrast, YASGUI provides a comprehensive variety of methods to explore the results.

3.2.2 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 using LD Browsers. LD Browsers allow the user to navigate and explore LD in a similar manner to browsing the web such as in Tabulator [22], Humboldt [77], Ozone browser [28], Parallax [69], gFacet [62] and Rhizomer [27].

As LD Browsers are generally concerned with data visualisation and navigation, they lack ability to form specific queries. Thus, answering questions from a knowledge graph that has millions of triples is tedious and time-consuming. However, some LD browsers offer advanced querying features as in Falcons Explorer (discussed in section 3.1.1).

3.2.3 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 [35] 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 [52]. Thus, it is suitable for only particular search applications that require basic text searches [30].

3.2.4 Question Answering

Question Answering systems will interpret the user's question in natural language and return the appropriate answer in the same natural language [66]. The user's primary interest is obtaining the correct answer. Thus, the user is not expected to understand the underlying structure of data nor the query language [111]. 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 [47] 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. Various other efforts have been made to achieve the same purpose, such as in Aqqu [19], NLQ/A [117], BELA [106], and SINA [107].

The RDF Data Cube Vocabulary [9] is used to represent statistical data-cubes which is multi-dimensional values in LD [64]. Answering questions over statistical data using this type of vocabulary has not been appropriately explored [37]. For example, what was the average monthly income for a UK football player in 2019? CubeQA [65], 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.

3.2.5 SPARQL to Natural Language

It can be challenging for a Lay-user to interpret what a particular SPARQL query means or fulfils. As discussed earlier, it is unrealistic to expect all users to be familiar with SPARQL. So, a more practical step is to convert queries to the user's natural language, allowing the user to comprehend the query efficiently. These tools can lower the cognitive load for understanding SPARQL queries even for experienced users [95, 114]. SPARQL2NL [94] and SPARTIQULATION [42] are examples of SPARQL to Natural Language translators.

SPARQL2NL can translate SPARQL queries to Natural Language based on the selected SPARQL endpoint. In addition, it allows the user to run the query and retrieve the results. Then, the tool explains the results in Natural Language. Thus, the user does not have to understand RDF or how resources are connected as the tool will explain it in a written format [94]. In contrast, SPARTIQULATION attempts to translate SPARQL into Natural Language using 'Document Structuring' by converting the entire query into a graph representation and then translating it into Natural Language. SPARTIQULATION thus lacks the flexibility to make changes based on the data structure, as some messages are hard-coded. Both tools have shown encouraging results despite requiring further improvements [94, 42].

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

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

4 Semantic Web Solutions

This section explores the current semantic web solutions by reviewing papers published as an in-use resource or application. The goal was to broaden the scope of the search in order to uncover new areas and prospective answers and ideas.

4.1 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. [14] 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. [43] 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.

4.2 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 [41] 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. [49] 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 [119] 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) [13] 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) [104] 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 [103], 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.

4.3 Mobile Applications

Nutrition and maintaining a healthy diet is one of the LD applications. Dragoni et al. [40] 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 Dragoni [38] 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. [40] 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. [13] proposed scale.

Additionally, Donadello and Dragoni [38] 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.

4.4 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 [10]. 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].

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

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 '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 [75] 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 [75] 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. [44] 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.

4.5 Web APIs

Web developers primarily rely on Web Application Programming Interfaces (APIs) to consume Web Services and retrieve data [115]. 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)[11] 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 [12], 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 [115, 90]. Therefore, Meroño-Peñuela and Hoekstra [90] introduced grlc [13], 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.

[11] https://www.openapis.org
[12] https://json-ld.org/
[13] https://grlc.io/

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 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 [48] 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.

5 Evaluation and Research Validation

5.1 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 [51]. According to the ISO 9241-11, the quantitative usability evaluation has to achieve satisfaction, efficiency and effectiveness with measurable characteristics [24]. 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 [14] 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) [60] 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 effectiveness of the information retrieval process.

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

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 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.

5.2 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

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. [14]	N/A						✓						
Farah et al. [43]	N/A						✓						
Donadello and Dragoni [38]	120	✓					✓				✓		
Dragoni et al. [40]	49	✓							✓				
Fernàndez-Cañellas et al. [44]	N/A						✓						
Lisena et al. [83]	55								✓		✓	✓	
Meroño-Peñuela and Hoekstra [90]	6								✓			✓	
Garijo and Osorio [48]	N/A								✓				
Kejriwal and Szekely [75]	8 SME	✓			✓						✓	✓	
Lasolle et al. [79]	1 SME		✓		✓		✓						
Şimşek et al. [119]	51	✓	✓								✓	✓	
Eldesouky et al. [41]	120	✓				✓	✓		✓		✓		
Osborne et al. [97]	8 SME	✓			✓						✓	✓	
Thanapalasingam et al. [110]	14 SME	✓			✓						✓	✓	
Salatino et al. [104]	9 SME	✓			✓		✓				✓	✓	
Beschi et al. [23]	21	✓	✓	✓							✓	✓	
Vega-Gorgojo et al. [113]	15		✓			✓	✓		✓				
Kuric et al. [78]	15	✓				✓			✓	✓	✓		
Haag et al. [55]	10									✓	✓	✓	
Hoefler et al. [63]	14			✓		✓				✓	✓	✓	
Haag et al. [58]	6								✓		✓	✓	
Vargas et al. [112]	28		✓	✓		✓			✓		✓	✓	
Soylu et al. [109]	10		✓			✓			✓	✓	✓		
Kaufmann et al. [73, 74]	48	✓	✓			✓			✓		✓	✓	
Zenz et al. [116]	-					✓			✓				
Ferré [45]	26	✓				✓			✓		✓	✓	

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 [32]. 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 [32, 6]. If the LD is periodically updated, an exact match for the output is impossible, as even the 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 [32, 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.

6 Findings and Discussion

This survey reviewed a wide range of available tools for accessing LD, with a focus on generating SPARQL, as this is the W3C recommended query language for LD. Most of these tools address a variety of users, yet many have some weaknesses in supporting certain user types, most commonly Lay-users. Experienced users are familiar with RDF and can understand most interfaces' metaphors clearly, while Lay-users struggle to deal with some of these tools as they require prior RDF knowledge [34, 36, 112, 2, 63, 78, 47]. Figure 8 summarises the main pros and cons of the key types of SPARQL QBs covered.

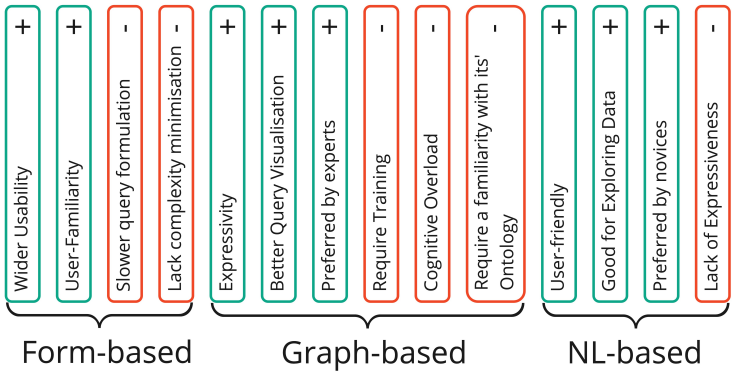

Figure 8: The summarisation of the findings by listing the Pros (+) and Cons (-) of the main types of SPARQL query builders.

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.

7 Research Challenges and Future Directions

Although the reviewed works and solutions have covered a wide range of domains, there are still some common gaps and areas that were not addressed. In addition, several researched tools were not evaluated or tested in terms of usability, necessitating further investigation into the usefulness of the proposed solutions. In terms of wider usability, none of the investigated tools supports the combination of basic/advanced UIs to lower barriers for Lay-users as a way to accelerate their interface adaptation while still supporting advanced querying for Experts. In addition, NL-Based Query Builders do not support the conversion of SPARQL to NL. For example, tools such as SPARQL2NL allow the user to convert SPARQL to NL, which is not supported by NL-Based QBs. Supporting such a feature can assist Lay-users in evaluating the generated SPARQL. Thus, this issue must be examined to determine whether such capability might improve the tool evaluation results.

Supporting Web of Things (WoT) and Spatial Data. The development of Smart homes and cities and the extension of WoT domains have introduced new concepts to traditional LD, including sensors, observations, and spatial data. Most of the reviewed query formulation tools were only concerned with extracting general knowledge from traditional LD. Even though these tools are designed to work with general LOD, they lack the simple concepts to access and explore LD that follows, for example, the specifications of "Sensor, Observation, Sample, and Actuator (SOSA)"[70] ontology. Reapplying with some of the reviewed techniques or considering a new visualisation approach to this type of LD is essential to the end-user.

Traditional LD will involve a simple RDF pattern such as 'Adam is Human'. However, when it comes to Smart buildings, the information is retrieved differently. For example, a thermal sensor may be positioned in an area where many other sensors exist, obtaining readings on a regular basis for various measurements. The sensor's readings will thus reveal a many-to-one relationship that connects numerous records to a single sensor. Despite these relationships being described in the ontology, the visualisation and relationships are more condensed and complicated. So, in terms of data querying to obtain information related to the building condition, the sensor is thus of more interest than the readings. For example, the users are not interested in a single reading corresponding to a simple RDF pattern; instead, they require the ability to query all data linked to a single sensor while filtering out undesired readings by matching a specific pattern.

Results Perception During Query Construction. Form-based QB seems to be the least intimidating interface among the other QBs for novel users; however, it requires improvement in terms of usability. FILTER/FLOW MODEL has shown that providing preliminary results during query construction helps users build accurate queries. None of the other query builders has supported such a feature with respect to reflecting the possible results during the query construction, so the use of this technique needs further investigation.

This is important due to the fact that, as we previously discussed, even experts might have ambiguity in their views about the expected results that may affect the query construction. Thus, offering some insight to the user about the potential result during construction might improve their overall experience. For example, showing the size of the results by increasing the visual model's thickness to symbolise increased result size might be helpful in terms of determining the usefulness of any used filters. Alternatively, the density of the result could also be represented by using darker colours to reflect the intensity of the data. If the results incorporate geospatial data, a map might also be a good means of visualisation, significantly influencing the user's perception of the data. In addition, when users visually select an entity, displaying all the associated entities might improve the query construction approach as compared to the user having to guess the entities. Therefore, the perception of the results is an important factor that may positively impact usability.

Supporting Conversational AI. NL and Keyword-based approaches are concerned with extracting knowledge from LD using a sentence or keywords. Thus, there was no potential strategy to detect the user intent and thus return feedback to improve their query construction. In contrast, a Virtual Assistant (VA) is an EUD tool that was not reviewed in the context of accessing LD. Barricelli et al. [16] discussed the possibility of using VA and Conversational AI in the domain of IoT to enable end-users to manage their IoT environment more easily. VA can be used to capture the user intent more accurately and present the results in the users' natural language. The current use of VAs is limited to finding a conclusion or using predefined SPARQL templates, which requires querying by replacing the entity values like in Mishra et al. [92] paper; however, no support for accessing the data using any adaptive or scalable approaches has been found.

Improving Usability by Integrating Multiple Tools. Each of the reviewed tools and visualisation approaches has pros and cons. Regardless, these tools did not examine the effectiveness of combining multiple approaches to address known flaws and improve tool usability. As an example, it is thus unknown whether combining Form-based tools with Graph-based tools would minimise the cognitive load seen in solely graph-based instances while improving expressivity, which is limited in the form-based approach. Similarly, it may be that merging the NL with a Form-based approach might improve the speed of query construction. Integrating multiple visualisation paradigms should be carefully tested to avoid overwhelming the user by increasing the tool's complexity, however, suggesting that additional functionality should be introduced as an assistive tool or an alternative visualisation that is tightly coupled to reflect the same query.

8 Conclusion

The Linked-Data (LD) paradigm has demonstrated remarkable potential of representing information and delivering human and machine-readable formats. However, its potential is being confined by the fact that its users are expected to have a prior understanding of the complex SPARQL query language to access the data. This paper has reviewed various tools and approaches to access LD, including analysing alternatives to manually writing SPARQL. We have categorised the tools based on the UI querying approach. In addition, the paper investigates the current solutions and their research validation techniques. The results showed weaknesses and a lack of adequately supporting a wide range of users, especially Lay-users.

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] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 3540762973. doi: 10.1007/978-3-540-76298-0_52.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] S. Beschi, D. Fogli, and F. Tampalini. CAPIRCI: A Multi-modal System for Collaborative Robot Programming. In A. Malizia, S. Valtolina, A. Morch, A. Serrano, and A. Stratton, editors, *End-User Development*, pages 51–66, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24781-2. doi: 10.1007/978-3-030-24781-2_4.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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/`.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

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

[70] 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`.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[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. 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.

[104] 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.

[105] 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.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

[114] 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.

[115] 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.

[116] 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.

[117] 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.

[118] 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.

[119] 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.