RDF Explorer: A Visual SPARQL Query Builder

Hernán Vargas[1,3], Carlos Buil-Aranda[1,3], Aidan Hogan[2,3], and Claudia López[1]

[1] Universidad Técnica Federico Santa María, Valparaíso, Chile
{hvargas,cbuil,claudia}@inf.utfsm.cl
[2] DCC, Universidad de Chile, Chile ahogan@dcc.uchile.cl
[3] Millenium Institute for Foundational Research on Data (IMFD), Chile

Abstract Despite the growing popularity of knowledge graphs for managing diverse data at large scale, users who wish to pose expressive queries against such graphs are often expected to know (i) how to formulate queries in a language such as SPARQL, and (ii) how entities of interest are described in the graph. In this paper we propose a language that relaxes these expectations; the language's operators are based on an interactive graph-based exploration that allows non-expert users to simultaneously navigate and query knowledge graphs; we compare the expressivity of this language with SPARQL. We then discuss an implementation of this language that we call RDF Explorer and discuss various desirable properties it has, such as avoiding interactions that lead to empty results. Through a user study over the Wikidata knowledge-graph, we show that users successfully complete more tasks with RDF Explorer than with the existing Wikidata Query Helper, while a usability questionnaire demonstrates that users generally prefer our tool and self-report lower levels of frustration and mental effort.

1 Introduction

Over the past decade, hundreds of datasets have been published using the Semantic Web standards covering a variety of domains [30]. These datasets are described using the RDF data model, which is based on graphs. Beyond the Semantic Web community, the idea of using graphs to model and manage diverse data at large-scale has also become increasingly popular, marked by the recent announcements of various *knowledge graphs* [12]. Some of these knowledge graphs are proprietary, maintained internally by companies such as Google, Microsoft, Apple, etc.; while others are open to the public via the Web, maintained by dedicated international communities, like DBpedia [22], Wikidata [36], etc.

A number of query languages have then been proposed specifically for graphs, including SPARQL for RDF graphs, Cypher for property graphs, etc. [3]. However, querying graphs using these languages can be challenging. First, users are required to have technical knowledge of such query languages and the semantics of their operators. Second, graphs are often used to represent diverse data that may not correspond to a particular domain-specific schema, meaning that the users may not be easily able to conceptualize the data that they are querying, particularly for domain-agnostic knowledge graphs. Despite

these limitations, query services for DBpedia and Wikidata are receiving in the order of millions of queries per day [28,23]; although many such queries are from "bots", tens of thousands are not [23], where such statistics indicate the value of being able to query graphs for many users and applications.

Several interfaces have been proposed to allow lay users to visualize, search, browse and query knowledge graphs, with varying goals, emphases and assumptions [15]. Some key approaches adopted by such interfaces (discussed further in Section 2) involve keyword search, faceted browsing, graph-based browsing, query building, graph summarization, visualization techniques, and combinations thereof. In general however, many proposed systems trade expressivity – the types of operators and interactions supported, and thus the types of queries that can ultimately be captured through the interface – for usability and efficiency. Few interfaces have been proposed, for example, that can handle graph-patterns with cycles, such as to find siblings who have directed movies together, drugs indicated and contraindicated for pairs of comorbid illnesses, pairs of binary stars of the same classification, and so forth. Interfaces that *can* capture such graph patterns often assume some technical expertise of the query language and/or knowledge of how data are modeled.

This work proposes a language and associated interface that enables lay users to build and execute graph-pattern queries on knowledge graphs, where the user navigates a visual representation of a sub-graph, and in so doing, incrementally builds a potentially complex (cyclical) graph pattern. More specifically, we first propose a set of operators, forming a language that allows users to build SPARQL graph patterns by interactively exploring an RDF graph; we further study the expressivity of this language. We then discuss the design of a user interface around this language, and the additional practical features it incorporates to improve usability, such as auto-completion, result previews, generalizing examples, etc.; we further describe how this interface can be implemented on top of an existing query service (SPARQL endpoint). Our claim is that the resulting interface allows lay users to express graph-pattern queries over knowledge graphs better than existing interfaces that support similar expressivity. To evaluate this claim, we present a task-based user study comparing the usability of our interface with the Wikidata Query Helper; the results indicate that users achieve a higher successful completion rate of tasks with our system.

2 Related Work

A wide variety of interfaces have been proposed in recent years to help lay users visualize and explore RDF graphs [15,11]. Amongst these works, we can first highlight search and browsing systems that allow users to find entities by keyword and potentially filter or modify results by selecting facets or following paths (e.g., *Tabulator* [9], *Explorator* [4], *VisiNav* [19], amongst others); these approaches are limited in terms of the types of queries that they can express, not allowing (for example) to express cycles. Other types of interfaces focus on providing visualizations to summarize data, be they domain-independent (e.g.,

Sgvizler [31],or domain-specific (e.g., *DBpedia Atlas* [35], *DBpedia Mobile* [8], *LinkedGeoData Browser* [34]) visualizations; such systems focus on providing overviews of data rather than exploring or querying for specific nodes/entities. Other systems combine browsing/exploration and visualization, often following a graph-based navigation paradigm (e.g., *RDF Visualiser* [29], *Fenfire* [20], etc.); these systems allow to focus on a specific node and explore its neighborhood in the graph, but do not allow to generalize these explorations into queries.

To help users express more complex forms of queries over graphs, various *query editors and builders* have been proposed for languages such as SPARQL [16]. We provide an overview of such systems with publications in Table 1. For space reasons we focus on features that relate to the present contribution, omitting, for example, discussion of reasoning support in systems such as *QueryVOWL* [17] and *OptiqueVQS* [33], or the schema-based notation used by *SPARQLing* [7]. Such interfaces must deal with two antagonistic goals: supporting complex queries while assuming as little technical expertise on the part of the user as possible. Towards the more expressive end of the scale are query editing interfaces – such as *SPARQL Assist* [24], *YASGUI* [27], etc. – which offer users some helpful features when formulating SPARQL queries in a text field, but still assume knowledge of SPARQL. On the other hand, query builders aim to abstract away from SPARQL syntax, allowing to formulate queries in a more visual way, based on form fields or graphs. From Table 1, we conclude that the closest system to ours is *Smeagol* [14], which also supports key features such as autocompletion, example-based querying (where users explore a graph and then generalize some constants as variables), dynamic results (where query results are generated on the fly and used to guide query construction), and non-emptiness guarantees (to avoid users generating queries with zero results); furthermore, *Smeagol* offers a task-driven user evaluation against a baseline *Pubby* system with a substantial number of users and significance testing. Our proposal is distinguished from *Smeagol* in key aspects; most importantly, while *Smeagol* [14] focuses on supporting tree-shaped queries generated during user exploration, our proposal also has general support for graph patterns.

Research on usable interfaces for querying graphs can not only have impact beyond the Semantic Web community, it can also benefit from expertise in other communities. In particular, the area of Human Computer Interaction (HCI) can offer insights not only in terms of the challenges faced in such research, but also in the design of user studies to evaluate the claims made of such research. Along these lines, Bhowmick et al. [10] reflect on recent advances in what they refer to as the *visual graph querying paradigm* from the HCI perspective, characterizing the challenges in the area and the research directions that should be followed. The authors define the challenges as follows: (1) the development of graph queries requires a considerable cognitive effort; (2) users need to be able to express their goal in a systematic and correct manner, which is antagonistic with the goal of catering to lay users; (3) it is more intuitive to "draw" graph queries than to write them, which implies the need for intuitive visual interfaces. Regarding the latter point, the authors claim that current visual querying tools

Table 1: Comparison of query interfaces for SPARQL, indicating the year of the associated publication, the mode of interaction, the features supported (AC = Autocomplete; EX = Example-based Querying; DY = Dynamic Results, NE = Non-Empty Results), details of user evaluation conducted, if any (B = Baseline, Q = Questionnaire, T = Tasks) and details of availability

System	Year	Mode	Features				User Eval.	Expressivity
			AC	EX	DY	NE		
NIGHTLIGHT [32]	2008	GRAPH	–	–	–	–	–	SPARQL 1.0^-
Konduit [2]	2010	FORM	✓	–	–	–	–	$BGPs^+$
RDF-GL [21]	2010	GRAPH	–	–	–	–	BQT (5 users)	SPARQL 1.0^-
Smeagol [14]	2011	GRAPH	✓	✓	✓	✓	BQT (43 users)	Trees
SPARQL Assist [24]	2012	TEXT	✓	–	–	–	–	$BGPs^+$
QUaTRO2 [6]	2013	FORM	✓	–	✓	✓	*no details*	$Trees^+$
QueryVOWL [17]	2015	GRAPH	✓	–	–	–	QT (6 users)	BGPs
YASGUI [27]	2017	TEXT	✓	–	–	–	–	SPARQL 1.1
OptiqueVQS [33]	2018	GRAPH	✓	–	–	–	T (10 users)	Trees
SPARQLing [7]	2018	GRAPH	–	–	–	–	–	Trees
ViziQuer [13]	2018	GRAPH	–	–	–	–	BT (14 users)	$Trees^+$
WQH [23]	2018	FORM	✓	–	✓	–	–	BGPs
RDF Explorer	2019	GRAPH	✓	✓	✓	✓	BQT (28 users)	$BGPs^-$

suffer from poor aesthetics. They further indicate important primitives that such tools should support to cater to diverse users and support diverse queries: *edge creation*, *pattern creation* and *example-based querying*. Aside from this, they emphasize *action-aware query processing* in which the system is able to deliver partial information and immediate feedback while the user is developing the query, based on *dynamic result exploration and visualization*. They acknowledge, however, that these goals, when taken together, are challenging to address given large-scale graphs and complex queries.

In this paper, we propose (yet another) visual query builder for SPARQL. In so doing, we are particularly inspired by the discussion of Bhowmick et al. [10] in terms of the main interactions and features that are key to making such systems usable for non-experts, and by the "specific-to-general" paradigm adopted by *Smeagol* [14]; however, we adopt various extensions to improve usability and expressivity, key among which is support for graph patterns with cycles.

3 RDF Explorer

In this section, we propose our RDF Explorer system, whose goal is to enable lay users to query and explore RDF graphs. We first discuss the operators that form the basis of a visual query language in which users can express queries over graphs through simple interactions; we characterize the expressivity of the language in relation to SPARQL. Thereafter, we discuss how this query language is supported by the RDF Explorer interface, and how the overall system is implemented.

3.1 Visual Query Graph

The visual query language we propose is formulated with respect to a visual query graph. Let $\mathbf{I}$ denote the set of IRIs, $\mathbf{L}$ denote the set of literals and $\mathbf{V}$ denote the set of query variables. We define the visual query graph as follows.

Definition 1. *A visual query graph (VQG) is defined as a directed, edge-labelled graph* $\mathsf{G} = (\mathsf{N}, \mathsf{E})$*, with nodes* N *and edges* E*. The nodes of the VQG are a finite set of IRIs, literals and/or variables:* $\mathsf{N} \subset \mathbf{I} \cup \mathbf{L} \cup \mathbf{V}$*. The edges of the VQG are a finite set of triples, where each triple indicates a directed edge between two nodes with a label taken from the set of IRIs or variables:* $\mathsf{E} \subset \mathsf{N} \times (\mathbf{I} \cup \mathbf{V}) \times \mathsf{N}$*.*

We denote by $\text{var}(\mathsf{G})$ the set of variables appearing in $\mathsf{G} = (\mathsf{N}, \mathsf{E})$, either as nodes or edge labels: $\text{var}(\mathsf{G}) \coloneqq \{v \in \mathbf{V} \mid v \in \mathsf{N} \text{ or } \exists\, n_1, n_2 : (n_1, v, n_2) \in \mathsf{E}\}$.

We say that the VQG is *constructed* through a *visual query language*, consisting of four algebraic operators that will correspond to atomic user interactions: adding a variable node, adding a constant node, adding an edge between two existing nodes with a variable label, and adding an edge between two existing nodes with an IRI label. More specifically, the VQG is initially empty: $\mathsf{G}_0 = (\emptyset, \emptyset)$. Thereafter, a VQG can be constructed through the visual query language (VQL), defined straightforwardly as follows.

Definition 2. *Letting* $\mathsf{G} = (\mathsf{N}, \mathsf{E})$ *denote the current VQG; the visual query language (VQL) is defined through the following four atomic operations:*

- *Initialize a new variable node:* $\eta(\mathsf{G}) \coloneqq (\mathsf{N} \cup \{v\}, \mathsf{E})$ *where* $v \notin \text{var}(\mathsf{G})$*.*
- *Add a new constant node:* $\eta(\mathsf{G}, x) \coloneqq (\mathsf{N} \cup \{x\}, \mathsf{E})$ *where* $x \in (\mathbf{I} \cup \mathbf{L})$*.*
- *Initialize a new edge between two nodes with a variable edge-label:* $\varepsilon(\mathsf{G}, n_1, n_2) \coloneqq (\mathsf{N}, \mathsf{E} \cup \{(n_1, v, n_2)\})$ *where* $\{n_1, n_2\} \subseteq \mathsf{N}$ *and* $v \notin \text{var}(\mathsf{G})$*.*
- *Add a new edge between two nodes with an IRI edge-label:* $\varepsilon(\mathsf{G}, n_1, x, n_2) \coloneqq (\mathsf{N}, \mathsf{E} \cup \{(n_1, x, n_2)\})$ *where* $\{n_1, n_2\} \subseteq \mathsf{N}$ *and* $x \in \mathbf{I}$*.*

Note that for the VQL operators $\eta(\mathsf{G})$ and $\varepsilon(\mathsf{G}, n_1, n_2)$, the variable is not specified, where rather an arbitrary fresh variable can be automatically generated. No matter what variables are chosen, since the variables added are always fresh, the resulting VQG will be unique modulo isomorphism; in practice, the system can thus take care of generating fresh names for each variable.

Though VQGs are a straightforward way to represent queries against graphs, since VQGs allow for representing cycles, they already go beyond the expressivity of many user interfaces for graphs (entity search, facets, etc.), and even many of the related visual query languages proposed in the literature, which are based on trees (see Table 1). On the other hand, we choose not to support query operators that go beyond simple graph patterns as covered by similar graph-based interfaces – such as NIGHTLIGHT [32] and RDF-GL [21], which support unions, optional, etc. – as we consider such systems to be aimed at users with some knowledge of query languages and do not know of an intuitive way to represent such operators in a manner that would be accessible to a lay user. On the other hand, VQGs will be converted to concrete SPARQL syntax, where a more expert user can modify the resulting query as required.

3.2 Translating VQGs to SPARQL

VQGs are designed as a visual metaphor for SPARQL basic graph patterns (BGPs), where the translation is thus *mostly* direct and natural; however there are BGPs that cannot be expressed as VQGs, and indeed, there are minor aspects of VQGs that cannot be translated to BGPs. Before we discuss such issues, we must first introduce some notation for RDF and SPARQL BGPs [26].

An *RDF triple* uses terms from the set of IRIs ($\mathbf{I}$), literals ($\mathbf{L}$) and blank nodes ($\mathbf{B}$); more specifically a triple $t = (s, p, o)$ is an RDF triple iff $s \in \mathbf{I} \cup \mathbf{B}$ (called the *subject*), $p \in \mathbf{I}$ (called the *predicate*) and $o \in \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$ (called the *object*). A finite set of RDF triples is called an *RDF graph*.

SPARQL *basic graph patterns* (BGPs) correspond to RDF graphs, but where variable terms ($\mathbf{V}$) can also be used. Along these lines, a triple $q = (s, p, o)$ is a SPARQL *triple pattern* iff $s \in \mathbf{I} \cup \mathbf{L} \cup \mathbf{V}$, $p \in \mathbf{I} \cup \mathbf{V}$ and $o \in \mathbf{I} \cup \mathbf{L} \cup \mathbf{V}$.[4] A SPARQL BGP is then a finite set of SPARQL triple patterns. The semantics of a BGP is defined in terms of its *evaluation* over an RDF graph, which returns a set of *mappings*. A mapping $\mu : \mathbf{V} \to (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is a partial map from variables to RDF terms; the set of variables for which μ is defined is called the *domain* of μ, denoted $\text{dom}(\mu)$. Given a query Q, we denote the set of variables it mentions by $\text{var}(Q)$; furthermore, we denote by $\mu(Q)$ the image of Q under μ: the result of replacing every occurrence in Q of every variable $v \in \text{var}(Q)$ by $\mu(v)$ (or v if $v \notin \text{dom}(v)$). The *evaluation* of a BGP Q with respect to an RDF graph G, denoted $Q(G)$, is then defined as the set of mappings $\{\mu \mid \text{dom}(\mu) = \text{var}(Q) \text{ and } \mu(Q) \subseteq G\}$ (note that this is equivalent to – but more succinct than – defining the evaluation of a BGP as a join of the evaluation of its constituent triple patterns).

In terms of translating VQGs to BGPs, given a VQG $\mathsf{G} = (\mathsf{N}, \mathsf{E})$, we observe that by design, the set E is already a BGP, and we are done. However, first we must remark that this translation is agnostic to *orphan nodes* – nodes with no incident edges – in G; thus for example, G and $\eta(\mathsf{G})$ will give the same BGP. Second, while VQGs are BGPs have equivalent definitions, not all VQGs/BGPs can be constructed by the four operators in the visual query language described earlier. In particular, we cannot construct VQGs/BGPs where a join variable – a variable appearing in more than one edge/triple pattern – appears as an edge-label/predicate (since the $\varepsilon(\mathsf{N})$ operation is defined only for fresh variables, while $\varepsilon(\mathsf{G}, n_1, x, n_2)$ is defined only where x is constant); we do not consider this to be an important limitation in practice since analysis of real-world SPARQL query logs suggests that joins on the predicate position are rare [5].

The VQG can then be serialized in concrete SPARQL syntax: the corresponding basic graph pattern is written as the `WHERE` clause of a SPARQL query, where all variables are projected with `SELECT *`; at this point, a more expert user may wish to modify the query, e.g., adding query operators.

With respect to complexity, we remark that for the evaluation decision problem – which asks: given a mapping μ, a query Q and an RDF graph G,

[4] We do not consider blank nodes in triple patterns, which can be modeled as unprojected (aka. non-distinguished) query variables.

is $\mu \in Q(G)$? – the queries generated by a VQG are tractable for this problem as they do not feature projection (a trivial upper bound is given by $O(|Q| \cdot |G|)$ [26]). However, in the interface we implement a number of features for usability, where one such feature is to suggest possible groundings of variables that will not lead to non-empty results. The corresponding decision problem for this autocompletion feature asks, given μ (where $\text{dom}(\mu) \subseteq \text{var}(Q)$), a query Q and an RDF graph G, is $\mu(Q)(G)$ non-empty? This problem is NP-complete in combined complexity (considering the size of G and Q in the input) since $\mu(Q)(G)$ can represent a graph, and one can reduce from the graph homomorphism problem; however, in data complexity (considering the query Q as fixed) the problem is tractable. In summary, the autocompletion task may become challenging as the VQG grows more complex; currently we rely on a SPARQL query to generate these suggestions, where we leave further optimizations for future work.

3.3 The RDF Explorer Interface

While the visual query graph offers a visual metaphor for basic graph patterns and the visual query language describes the interactions by which a visual query graph can be constructed incrementally by the user, these concepts leave many questions open regarding usability. One key issue, for example, is how the VQG should be visualized. Another practical issue we glossed over is that while $\eta(\mathsf{G})$ and $\varepsilon(\mathsf{G}, n_1, n_2)$ do not require any specific knowledge (in the latter, the user can select two nodes displayed in the visualization, for example), the operations $\eta(\mathsf{G}, x)$ and $\varepsilon(\mathsf{G}, n_1, n_2, x)$ require the user to give a specific (IRI or literal) term x, which assumes domain knowledge. Furthermore, we have yet to address the usability features discussed by Bhowmick et al. [10], such as *example-based querying*, *action-aware query processing*, or *dynamic result exploration and visualization*. Addressing such issues is key to achieving our goal of enabling lay users to formulate expressive queries over graphs. Along these lines, we now describe the RDF Explorer interface, which we propose to address these concerns.

The RDF Explorer interface is composed of six main components displayed in three panes. Figure 1 provides a screenshot of the interface for querying Wikidata, where we can see three components: a search panel (left pane), a visual query editor (center pane), and a node detail view (right pane); in the top right corner are buttons to switch the right pane to display one of the three other components: a node editor (allowing to add restrictions to a highlighted node), a SPARQL query editor (showing the current query), and a help panel.

The process starts with a blank visual query editor. The user must then start by adding a new node, be it a variable node ($\eta(\mathsf{G})$) or a constant node ($\eta(\mathsf{G}, x)$); for selecting x, the user can type a keyword phrase into the search pane on the left, which will generate autosuggestions, where any of the results shown can be dragged into the central query editor pane. The user may then proceed to add a second node by the same means. With two or more nodes available, the user can now click and drag between two nodes to generate an edge with a variable edge-label (shown as a box nested inside the source node); a list of potential

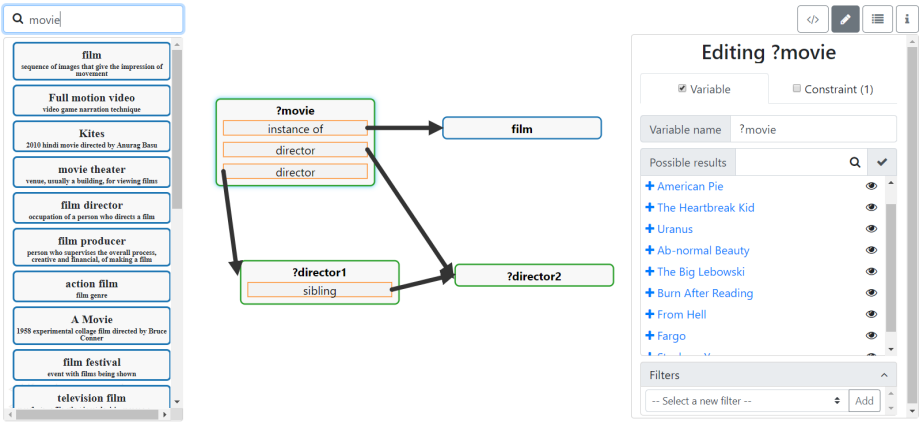


Figure 1: Example visual query finding siblings who have directed movies together

IRIs will be suggested for replacing the variable, where only IRIs that generate non-empty results for the underlying query will be offered.

Figure 2 illustrates some further features of the interface. Following conventions used in the case of property graphs [3], we display datatype properties within a given node to avoid clutter; this can be seen for the number of children property in Figure 2. At any point, the user may click on a node to view further details: if the node is variable (see Figure 1), they will be shown a sample of current results for that variable (generated by mapping the current VQG to SPARQL and projecting that variable); if the node is constant (see Figure 2), they will be shown the data available for that node, organized by datatype properties (which take a literal value) and object properties (which take an IRI value). In this way, per the discussion of Bhowmick et al. [10], the user can explore the graph and receive feedback on the results generated thus far, guiding next steps. Constant nodes can be converted to variables nodes, enabling the user to start with a specific example and then generalize the graph [14]. We claim that these features improve the usability of the system for lay users.

4 User Study

We now describe a user study that we conducted to evaluate our interface. We first make explicit our hypotheses and then describe the dataset, baseline system and user-study design that we selected to test these hypotheses. We then give details on the participants of the study and the metrics we collect.

Hypotheses The goal of our work is to enable users without prior knowledge of the Semantic Web to explore an RDF database and correctly build SPARQL

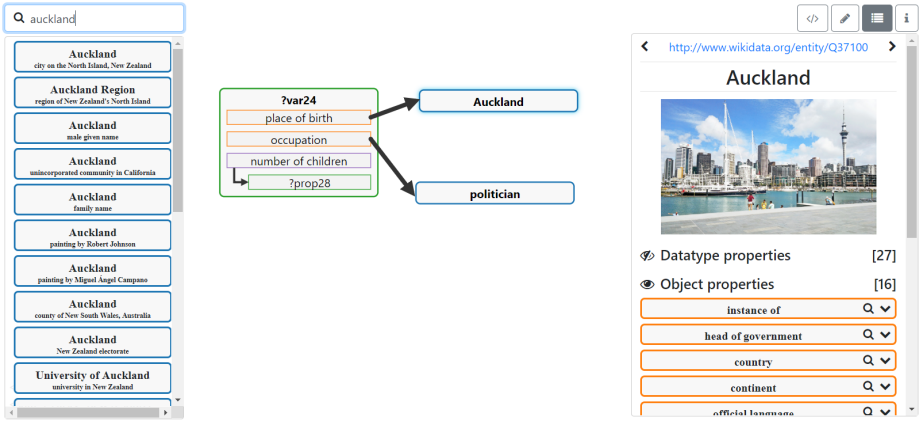


Figure 2: Example visual query finding politicians born in Auckland

queries encoding a specific information need. Our hypotheses are as follows, where each hypothesis relates to user success at different levels of granularity:

H_1: *Non-expert users are able to correctly formulate more SPARQL queries with our visual query builder than a baseline system.* For a query to be considered correct, it must return the same results as the reference query for the task.

H_2: *Non-expert users are able to correctly formulate more triple patterns with our visual query builder than a baseline system.* For a triple pattern to be considered correct in the generated query, it must be contained in the reference query (modulo variable names).

H_3: *Non-expert users are able to generate more correct query graphs with our visual query builder than a baseline system.* For a query graph to be considered correct, its "shape" must be the same as that of the reference query graph, irrespective of edge labels or node types/values. More formally, given a visual query graph $\mathsf{G} = (\mathsf{N}, \mathsf{E})$, let shape($\mathsf{G}$) denote a directed graph $S = (V_S, E_S)$ such that $V_S = \mathsf{N}$ and $(x, y) \in S$ if and only if there exists an edge-label l such that $(x, l, y,) \in \mathsf{E}$; now given the reference query graph G', a user's query graph G'', and their corresponding shapes $S' = \text{shape}(\mathsf{G}')$ and $S'' = \text{shape}(\mathsf{G}'')$, the user's query graph G'' is considered correct if and only if there exists an isomorphism $h : V''_S \to V'_S$ such that $h(S'') = S'$.

Dataset and baseline According to statistics recently published by Malyshev et al. [23], the *Wikidata Query Service*[5] receives millions of SPARQL queries per day, where tens of thousands of these queries are "*organic*" (written and posed by humans rather than bots). The Wikidata knowledge graph itself is a large, diverse graph, where at the time of writing it described 56,097,884 items and was being collaboratively edited by 21,049 active users; such a graph is unfeasible for

[5] `http://query.wikidata.org/`

Table 2: Two sets of five increasingly-complex tasks

№	Set 1 (S1)	Set 2 (S2)
1	Find all dogs	Find all actors
2	Find all popes who are female	Find all German soccer players who participated in FIFA 2014
3	Find all mountains located in European countries	Find all container ships located in European countries
4	Find all emperors whose father is also an emperor	Find all physicists whose spouse is also a physicist
5	Find all Nobel prize winners with a student who won the same Nobel prize	Find all participants of an Olympic sport with a relative who participates in the same sport

any user to conceptualize in its entirety. We thus view Wikidata as a potentially challenging use-case for our visual query builder and adopt it for our study.

In fact, Wikidata already has a default query builder deployed to help users query the knowledge graph: the *Wikidata Query Helper* (WQH) [23]. The WQH visual interface accompanies a text field displaying the current SPARQL query; changes in WQH are reflected in the query and vice versa. WQH is based on two main functionalities: the ability to define a *filter* that allows to select a property p and an object o, and the ability to *show* more data than what is being filtered by fixing a property value and adding a variable to its associated o. To help users select a given value for p and/or o, a search field is provided that autocompletes a keyword query and provides ranked suggestions to the user.

Study design To test the hypotheses, we design a task-based user study to compare the subjects' ability to solve tasks on the proposed interface versus the baseline interface [25]. This comparison focuses on the users' ability to perform query-based tasks, including aspects such as the users' performance overall, their perceived cognitive load, and usability aspects. Given limitations to how many subjects we could recruit, we use a within-subject design where each participant completes five tasks using our query builder and five similar tasks with the baseline. Each task consists of answering a question (given in natural language) that requires formulating a query to retrieve answer(s) from the Wikidata graph.

We divide the subjects into two groups. The first group is asked to build a set of five queries (S1) using the proposed interface; afterwards they are asked to build a different set of five queries (S2) using the baseline. Conversely, the second group is asked to build the first set of queries (S1) using the baseline and the second set (S2) with the proposed interface. This design controls for individual differences with participants using both interfaces. Counterbalancing the order of interfaces helps control for carry-over effects, such as learning or fatigue.

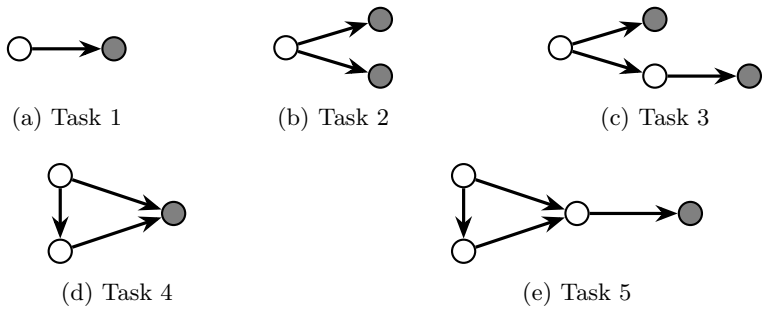
Table 3: Basic graph patterns corresponding to tasks listed in Table 2

№	Set 1 (S1)	Set 2 (S2)
1	`?dog wdt:P31 wd:Q144 .`	`?actor wdt:P106 wd:Q33999.`
2	`?pope wdt:P21 wd:Q6581072 .` `?pope wdt:P39 wd:Q19546 .`	`?ppl wdt:P1344 wd:Q79859.` `?ppl wdt:P27 wd:Q298.`

3	`?mount wdt:P31 wd:Q8502 .`
	`?mount wdt:P17 ?country .`
	`?country wdt:P30 wd:Q18 .`
		`?ship wdt:P31 wd:Q17210 .`
		`?ship wdt:P17 ?country .`
		`?country wdt:P30 wd:Q46 .`

4	`?emp1 wdt:P39 wd:Q39018 .`
	`?emp2 wdt:P39 wd:Q39018 .`
	`?emp1 wdt:P22 ?emp2 .`
		`?phy1 wdt:P106 wd:Q169470 .`
		`?phy2 wdt:P106 wd:Q169470 .`
		`?phy1 wdt:P26 ?phy2 .`

5	`?winner wdt:P166 ?novel .`
	`?student wdt:P166 ?novel .`
	`?student wdt:P802 ?winner .`
	`?novel wdt:P31 wd:Q7191 .`
		`?ppl1 wdt:P641 ?sp .`
		`?ppl2 wdt:P641 ?sp .`
		`?ppl1 wdt:P1038 ?ppl2 .`
		`?sp wdt:P279 wd:Q212434 .`



(a) Task 1 (b) Task 2 (c) Task 3

(d) Task 4 (e) Task 5

Figure 3: Expected query shapes for each pair of tasks shown in Table 2 where constant (IRI or literal) nodes are shaded and variable nodes are unshaded

Table 2 list the tasks in sets S1 and S2. The queries in both sets are designed to be of increasingly difficulty to follow a learning curve and also to avoid users being discouraged early on. We further aim to keep the n^{th} query of both sets of tasks comparable in terms of difficulty. Along these lines, as shown in Figure 3, each pair of tasks corresponds to the same abstract visual query graph, and each successive pair incrementally adds more complexity. Table 3 lists the target SPARQL basic graph patterns corresponding to each task in Table 2.

Before presenting each set of five tasks, we provide some brief training for the participants on how to use the interface they are about to see. Our training involves a description of the system's main functionalities and formulating an example query using these functionalities. For example, we pose the task *"Find all Clint Eastwood movies in which any of his children participated"* and show

how to build the query in the corresponding interface. A web page with other example queries for the interface is also provided to the participants.

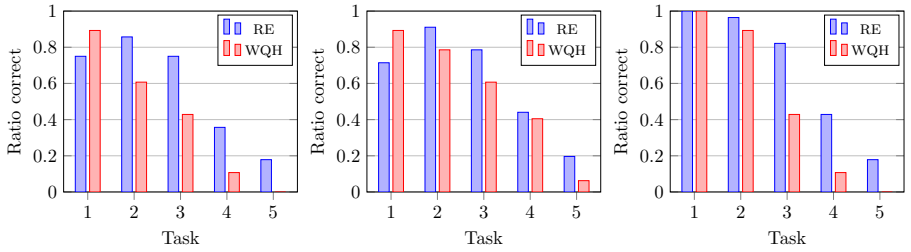
Study participants The study was conducted with 28 students enrolled in the undergraduate course "User interface design". The students were in the fourth year of a Computer Science undergraduate program in a Spanish-speaking university. They have no previous knowledge of SPARQL nor the Semantic Web. Their native language was Spanish; the text of tasks was presented in Spanish and while both interfaces were offered in English, a tooltip was added that automatically translates an English word to Spanish when the user hovers the mouse over a word. Participants were given up to 40 minutes to solve each of the sets of five tasks using each interface; adding two 5 minute tutorials before both sets of tasks, the total study time was thus 90 minutes.

Metrics To compare our visual query builder and the the baseline WQH query builder, we measure diverse aspects of the users' ability to perform a set of requested five tasks using each interface. We collect metrics for the users' task performance such as task completion rate and time for task completion. In terms of level of completion, we check the correctness of the query, the triple patterns, and the query graph (as previously described for our hypotheses). We also use the NASA Task Load Index [18] (NASA-TLX) to allow users to self-report the level of workload perceived by the user in a scale from 0 to 100. We use Likert scales from 1 to 5 to ask for usability aspects. We also include open questions to describe the data structure that the users believe to be behind each interface. We ask users to answer such questions using simple natural language that avoids technical jargon where we restrict the words they can use to the 1,000 most common words in their native language; we also ask them to illustrate (draw) how they understand the data structure.

5 Results

We begin by presenting the ratio of correct responses broken down by three levels of granularity: queries, triple patterns and shapes. Next we evaluate our hypotheses with respect to these data. We then present analysis of the subjective impressions of the interfaces.

Ratio of correct responses Figure 4 shows the mean ratios of correct responses for the proposed interface (RDF Explorer = RE) and the baseline interface (Wikidata Query Helper = WQH) at three different levels of granularity: queries, triple patterns, and shapes. Note that while queries and shapes are binary – either correct or not – in the case of triple patterns, we take the ratio of correct triple patterns versus total triple patterns provided in the response, where the presented results are then the mean of these ratios across all users.



(a) Queries (b) Triple patterns (c) Shapes

Figure 4: Mean ratios of correct results at three levels of granularity

Though we note a relatively high correctness ratio for earlier tasks, most users still struggled with later tasks; not only were earlier tasks easier, given the fixed time period for the study, some users did not reach the final task(s). Contrasting the two systems, in the first task, although all users of both systems got the query shape correct in both systems (which is trivially a single edge), they had more success correctly finding the terms of the triple pattern in WQH than RE; we believe that this is because WQH offers autocomplete forms that directly correspond to triple patterns whereas RE is more complex to use at first.[6] However, as tasks progress and queries become more "graph-like", users have more correct responses using the RE interface than the WQH interface; another possible interpretation is that users learn more about RE as the task progresses. Comparing the three levels of granularity, in the RE system, users generally have more success defining the correct query graph shape than identifying the terms (constants) in the query graph; the opposite trend is true for WQH, where users can more easily find the correct query terms, but not the correct query shape; we attribute this to two possible factors: the fact that WQH is form-based while RE is graph-based, and also based on the fact that RE blocks users from creating query shapes that give empty results while WQH does not.

Hypothesis testing To test our hypotheses, we assess the difference between completion rates of participants using both tools (whose mean values are depicted in Figure 4). We use *paired-t* tests to assess differences in the users' ability to perform the requested tasks; this test is appropriate because we are comparing the same participants using two different tools. We use $\alpha = 0.05$ to reject the null hypothesis and thus interpret that we have obtained a significant result when t^* is greater than $t_{crit} = 2.052$ ($t^* \geq t_{crit}$).[7] For our three hypotheses (see Section 4) the null hypotheses are that there is no difference between the tools or WQH performs better. The alternative hypothesis is that RE performs

[6] Given that the first task results in a query with a single triple pattern, the results for queries and triple patterns are the same.

[7] The value for t_{crit} is given by α and the number of participants ($n = 28$, giving $n-1 = 27$ degrees of freedom). See `http://www.numeracy-bank.net/?q=t/stt/ptt/3`.

better. We denote the completion rates for RE as $\bar{x}$ and those for WQH as $\bar{y}$; the average distances we denote by $\bar{d} = \overline{(\bar{x} - \bar{y})}$, and the standard deviation by s_d. We can then test the three hypotheses:

H_1 : *Non-expert users are able to correctly formulate more SPARQL queries with our visual query builder than a baseline interface.* We use the data summarized in Figure 4a. With $\bar{d} = \overline{(\bar{x} - \bar{y})} = 0.1714$ and $s_d = 0.2813$ we obtain $t^* = 3.22 > t_{crit} = 2.052$ rejecting the null hypothesis; i.e., we validate H_1 by obtaining a statistically significant result in favor of our interface.[8]

H_2 : *Non-expert users are able to correctly formulate more triple patterns with our visual query builder than a baseline interface.* We use the data summarized in Figure 4b. With $\bar{d} = 0.06$ and $s_d = 0.2609$ we obtain $t^* = 1.1947 < t_{crit} = 2.052$: the results are not statistically significant.

H_3 : *Non-expert users are able to generate more correct query graphs with our visual query builder than a baseline interface.* Here we use the data summarized in Figure 4c. With $\bar{d} = 0.1928$ and $s_d = 0.2801$ we obtain $t^* = 3.6431 > t_{crit} = 2.052$ rejecting the null hypothesis; i.e., we validate H_3 by obtaining a statistically significant result in favor of our interface.

Our user study is thus conclusive regarding the claim that our proposed interface is better than the baseline at helping non-expert users formulate their queries as graphs, but is not conclusive regarding the claim of our interface being better at helping users to correctly generate triple patterns.

Time results For space reasons, we present the results regarding task completion time as online reference data [1]. In summary, the average time needed for completing all ten tasks was 65 minutes while the fastest participant needed 50 minutes to complete all tasks.

Subjective results Figure 5 shows the results of the NASA-TLX questionnaire, where lower scores are deemed better. We see that for both systems, users still expressed concerns about both systems, where they found WQH particularly frustrating and demanding of mental effort; on the other hand, they found the the physical effort required to use WQH to be lower (perhaps because RE requires more clicks, drags, etc.). Figure 6 shows the usability results, where higher scores are better; the users express a preference across all dimensions for RE when compared with WQH.

6 Conclusions

We present a language and its visual implementation (RE) to support non-expert users in generating graph queries. Our results indicate that our RE interface

[8] The data were found to be normally distributed and there were no clear outliers; hence use of the paired t-test is considered valid. We also conducted a non-parametric Wilcoxon test to compare the users' ability to perform the requested tasks using the different interfaces; the results give a p-value of $0.001647 < 0.05$.

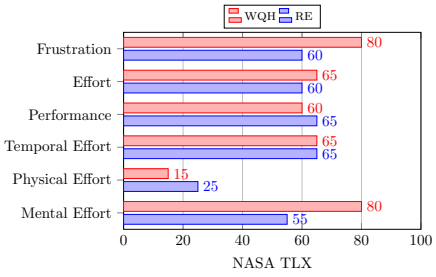


Figure 5: NASA-TLX results

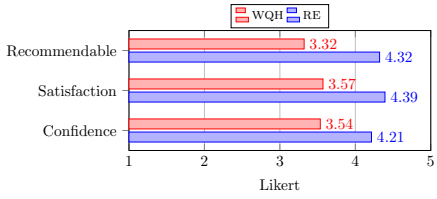


Figure 6: Likert results

is more effective at supporting non-expert users in creating correct SPARQL queries than the baseline WQH system. The data suggest that this difference could be attributed to better support for generating the correct graph patterns rather than the correct triple patterns, as well as usability features such as non-empty suggestions. Even though these benefits come at the cost of higher physical effort, they require lower mental effort and generate less frustration.

Future work can be followed along several lines. First, additional user studies may reveal further insights into the RDF Explorer system, where it would be of interest to compare with other baseline systems (such as Smeagol), with other endpoints, with other types of questions, and with a more diverse set of users. More generally, we could explore the potential reasons behind these usability differences, including whether or not our graph-like visualization leads users to develop more productive mental representations of the data structures. Aside from evaluation, the system could be improved along a number of lines, most importantly in terms of approximations for non-empty suggestions to improve performance for more complex visual query graphs, and support for features such as optionals, unions, etc. (while maintaining the usability of the system).

A demo of RDF Explorer is available at `https://www.rdfexplorer.org/` operating over the Wikidata SPARQL Query Service.

Acknowledgments Vargas and Buil-Aranda were supported by Fondecyt Iniciación Grant No. 11170714. Hogan was supported by Fondecyt Grant No. 1181896. Vargas, Buil-Aranda and Hogan were supported by the Millennium Institute for Foundational Research on Data (IMFD).

References

1. Online data:. In *URL* `http://www.rdfexplorer.org/data`.
2. O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop. In *Visual Interfaces to the Social and Semantic Web (VISSW)*. ACM Press, 2010.
3. R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, and D. Vrgoc. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.

4. S. Araujo, D. Schwabe, and S. Barbosa. Experimenting with Explorator: a direct manipulation generic RDF browser and querying tool. *Visual Interfaces to the Social and the Semantic Web (VISSW 2009), Sanibel Island, Florida*, 2009.
5. M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An Empirical Study of Real-World SPARQL Queries. In *Usage Analysis and the Web of Data (USEWOD)*, 2011.
6. B. Balis, T. Grabiec, and M. Bubak. Domain-Driven Visual Query Formulation over RDF Data Sets. In *Parallel Processing and Applied Mathematics (PPAM)*, pages 293–301. Springer, 2013.
7. S. D. Bartolomeo, G. Pepe, D. F. Savo, and V. Santarelli. Sparqling: Painlessly Drawing SPARQL Queries over Graphol Ontologies. In *International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA)*, pages 64–69, 2018.
8. C. Becker and C. Bizer. Exploring the geospatial semantic web with dbpedia mobile. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):278–286, 2009.
9. 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 *Proceedings of the 3rd international semantic web user interaction workshop*, volume 2006, page 159. Citeseer, 2006.
10. S. S. Bhowmick, B. Choi, and C. Li. Graph querying meets HCI: State of the art and future directions. In *ACM International Conference on Management of Data*, pages 1731–1736. ACM, 2017.
11. N. Bikakis and T. Sellis. Exploration and visualization in the web of big linked data: A survey of the state of the art. *arXiv preprint arXiv:1601.08059*, 2016.
12. P. A. Bonatti, S. Decker, A. Polleres, and V. Presutti. Knowledge graphs: New directions for knowledge representation on the semantic web. *Dagstuhl Reports*, 8(9):29–111, 2018.
13. K. Cerans, A. Sostaks, U. Bojars, J. Ovcinnikova, L. Lace, M. Grasmanis, A. Romane, A. Sprogis, and J. Barzdins. ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data. In *European Semantic Web Conference (ESWC)*, pages 158–163, 2018.
14. A. Clemmer and S. Davies. Smeagol: a "specific-to-general" semantic web query interface paradigm for novices. In *Database and Expert Systems Applications (DEXA)*, pages 288–302. Springer, 2011.
15. A.-S. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2):89–124, 2011.
16. P. Grafkin, M. Mironov, M. Fellmann, B. Lantow, K. Sandkuhl, and A. V. Smirnov. Sparql query builders: Overview and comparison. In *BIR Workshops*, 2016.
17. F. Haag, S. Lohmann, S. Siek, and T. Ertl. QueryVOWL: A Visual Query Notation for Linked Data. In *Extended Semantic Web Conference (ESWC)*, pages 387–402. Springer, 2015.
18. S. G. Hart and L. E. Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.
19. A. Harth. Visinav: A system for visual search and navigation on web data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):348–354, 2010.
20. T. Hastrup, R. Cyganiak, and U. Bojars. Browsing Linked Data with Fenfire. 2008.

21. F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*, pages 87–116. 2010.
22. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.
23. 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 *International Semantic Web Conference (ISWC)*, pages 376–394. Springer, 2018.
24. E. L. McCarthy, B. P. Vandervalk, and M. Wilkinson. SPARQL Assist language-neutral query composer. *BMC Bioinformatics*, 13(S-1):S2, 2012.
25. T. Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.
26. J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.
27. L. Rietveld and R. Hoekstra. The YASGUI family of SPARQL clients. *Semantic Web*, 8(3):373–383, 2017.
28. M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A. N. Ngomo. LSQ: The Linked SPARQL Queries Dataset. In *International Semantic Web Conference (ISWC)*, pages 261–269. Springer, 2015.
29. C. Sayers. Node-centric rdf graph visualization. *Mobile and Media Systems Laboratory, HP Labs*, 2004.
30. M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the Linked Data Best Practices in Different Topical Domains. In *International Semantic Web Conference, (ISWC)*, pages 245–260. Springer, 2014.
31. M. G. Skjæveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *Extended Semantic Web Conference*, pages 361–365. Springer, 2012.
32. 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. In *Knowledge Engineering and Knowledge Management (EKAW)*, pages 275–291. Springer, 2008.
33. A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-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.
34. C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.
35. F. Valsecchi, M. Abrate, C. Bacciu, M. Tesconi, and A. Marchetti. DBpedia Atlas: Mapping the Uncharted Lands of Linked Data. In *LDOW@ WWW*, 2015.
36. D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.