

KGSnap! in Practice: a Block-based Programming Environment for Enabling Knowledge Graph Literacy

Alessia Antelmi^{1,†}, Vincenzo Offertucci² and Maria Angela Pellegrino^{2,*}

¹Università degli Studi di Torino, Via Giuseppe Verdi, 8, 10124 Torino (Torino), ITALY

²Università degli Studi di Salerno, via Giovanni Paolo II, 132, 84084 Fisciano (Salerno), ITALY

Abstract

The growing availability of (linked) open data requires lay users to master how to deal with data effectively, yet SPARQL presents a barrier to leveraging data represented as knowledge graphs. As the block programming paradigm has been successfully used to teach programming skills, we demonstrate how to use KGSnap!, an extension of the block-based programming environment Snap!, to foster knowledge graph literacy among individuals lacking expertise in query languages. This work mainly focuses on the visualization and interaction aspects of KGSnap!, a visual SPARQL query builder, when experienced by users without expertise in the Semantic Web technologies. The reported experience is discussed as a learning-by-doing protocol aimed at facilitating the reproducibility and transparency of the performed evaluation. KGSnap! ease of use has been verified by 14 Snap! experts and 24 high-school learners. The findings indicate that lay users perceived it as a promising approach to acquaint themselves with knowledge graphs.

Keywords

SPARQL, Block-based programming paradigm, Snap!, Easy of use, Query-builder, Reproducibility

1. Introduction

Knowledge Graphs (KGs) have emerged as notable tools in enhancing educational processes and outcomes [1]. Significant effort has been dedicated to adhering to Linked Data (LD) principles in sharing educational data [2]. However, KGs remain largely underexplored as a learning objective [3, 4, 5, 6, 7]. Enabling KG learning requires investigating approaches that make them easily queryable by users lacking technical proficiency in Semantic Web technologies.


Towards this direction, block programming languages have emerged as a popular approach to introducing coding to non-experts [8]. Block-based environments enhance learnability for beginners by emphasizing recognition over recall, thus reducing cognitive load. This goal is achieved by representing computational patterns as blocks, allowing users to manipulate these blocks directly by dragging and connecting them like jigsaw puzzle pieces, thus preventing


VOILA 2024: The 9th International Workshop on the Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs co-located with the 23rd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 11-15, 2024.

*Corresponding author.

†These authors contributed equally to the authoring and refinement of the article.

✉ alessia.antelmi@unito.it (A. Antelmi); v.offertucci@studenti.unisa.it (V. Offertucci); mapellegrino@unisa.it (M. A. Pellegrino)

 0000-0002-6366-0546 (A. Antelmi); 0000-0001-8927-5833 (M. A. Pellegrino)

 © 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

errors and enhancing understanding of program structure [8]. Block-based programming environments aim to mask the complexity of querying KGs via SPARQL¹, recognized to be demanding for lay users [9, 10, 11].

This article describes the platform KGSnap!, an extension of Snap!² which follows the trend of supporting lay users to build and run queries on a working and publicly available SPARQL endpoint without previous knowledge of the syntax of SPARQL through a block-based programming approach. This work extends a previous article outlining the KGSnap! interface and interaction model [12] by answering to the following research question (RQ): *Is KGSnap! considered easy to use by lay users?* We assessed it through two distinct groups: (i) 14 experts proficient in Snap! programming to gauge their acceptance of integrating data literacy blocks, and (ii) 24 high school learners participating in intensive data literacy workshops to evaluate their proficiency in composing queries of increasing complexity. The results indicate that KGSnap! facilitates an intuitive interaction, enabling non-experts to engage with KGs.

The article is structured as follows. Section 2 overviews related work. Section 3 details KGSnap! interface and interaction aspects. Section 4 presents and discuss the ease of use results. Section 5 concludes the article with final remarks and future directions.

2. Related work

This section reviews visual SPARQL query builders and block-based programming environments. Table 1 provides a comparative analysis of visual SPARQL query builders developed within the last decade and block-programming environments, replacing the column ‘Approach’ with the intended objective of each tool.

Visual SPARQL query builders. Over the past years, different visual query builders have been proposed to hide the complexity of SPARQL and facilitate query construction. A basic distinction can be drawn between tools that require the usage of SPARQL syntax and those that opt for more intuitive interaction, thereby lowering the learning curve. Focusing on interfaces that mask SPARQL complexity, a plethora of different approaches have been proposed within the last decade. Some examples follow: queries can be modeled as data flows, as in SPARQL Filter Flow [13]; as graphs as implemented in QueryVOWL [14], RDF Explorer [9], Simplod [21]; queries can be formulated by requiring users interacting with forms as implemented in Spar-natural [24] and Wikidata Query Service [18]; with mashup, as in OptiqueVQS [16]; with conversational agents, as in Pellegrino et al. [19] and Forest QB [22]; or via a combination of controlled natural language and facets, as supported by Sparklis [15] and QueDI [20]. As a general trend, all the cited tools do not require users to explicitly deal with URIs.

Block programming environments. In the realm of block-based programming environments for LD, we can name several notable examples that go from supporting lay users in defining LD to query KGs. In the former category, Juma Uplift [27], extends Blockly to facilitate the definition of LD mappings. In the same vein, Sanctorum et al. [29] and Öztürk and Özacar [28] contributed to assisting non-expert users in ontology authoring and KG population. Toward the second direction, Punya [30] offers a block-based programming environment specifically

¹SPARQL: <https://www.w3.org/TR/sparql11-query>

²Snap!: <https://snap.berkeley.edu>

Table 1

Comparison of works similar to KGSnap!, considering their objective (i.e., SPARQL query builders) or their approach (i.e., adoption of the jigsaw metaphor). Legend: OK stands for full support, ~ for partial support, - for not supported or not explicitly reported, blanks mean not applicable.

Tool [Ref]	Approach / Objective	Masked URIs	Masked SPARQL	Material 4Edu
Visual SPARQL Query Builders				
SPARQL Filter Flow [13]	Data flow	OK	OK	-
QueryVOWL [14]	Graph	OK	OK	-
Sparklis [15]	Controlled NL & facets	OK	OK	~
OptiqueVQS [16]	UI mashup	OK	OK	~
SPARQLVis [17]	Graph	OK	OK	-
Wikidata Query Service [18]	Form-based	OK	OK	~
RDF Explorer [9]	Graph	OK	OK	-
Pellegrino et al. [19]	Conversational AI	OK	OK	~
QueDI [20]	Controlled NL & facets	OK	OK	~
Simplod [21]	Graph	OK	OK	~
Forest QB [22]	Conversational AI	OK	OK	-
KG VQL [23]	Query by example & Graph	OK	OK	-
Sparnatural [24]	Form-based	OK	OK	~
Block-based Interfaces				
SPARQL/CQELS [25]	<i>Query building</i>	-	OK	-
SparqlBlocks [26]	<i>Query building</i>	OK	OK	~
Juma Uplift [27]	<i>LD mappings definition</i>			-
Öztürk and Özacar [28]	<i>Ontology instantiating</i>			-
Sanctorum et al. [29]	<i>KG construction</i>			-
Punya [30]	<i>Query execution</i>			OK
KGSnap! [12, 31]	<i>Query building</i>	OK	OK	OK

tailored for SPARQL query execution and result handling. Similarly, SPARQL/CQELS Visual Editor [25] and SparqlBlocks [26] empower end-users to author SPARQL queries.

While SPARQL/CQELS Visual Editor does not mask URIs, KGSnap! and SparqlBlocks hide both URIs and the SPARQL language. One significant difference between those tools is that KGSnap! extends Snap!, whereas SparqlBlocks extends Blockly. Beyond being a technical distinction, Blockly enables developers to modify all category content. This approach results in environments that may appear similar but may offer different functionalities, which must be verified in any extension. Our proposed Snap! extension retains familiar blocks in their original positions, while new blocks are confined to a specific category. This approach could benefit the Snap! community that can extend their environment while maintaining traditional interaction with existing blocks. Notably, only Punya among block-based interfaces shares the philosophy we promote, i.e., supporting a structured learning experience with freely available material.

3. KGSnap!: querying Knowledge Graphs with Snap!

KGSnap!³ mirrors the structure of SPARQL queries, aligning with the philosophy of block programming to guide learners in gradually experimenting with the underlying language. The tool facilitates the formulation of SPARQL queries by specifying triples (subject, predicate, object), and covers basic graph patterns, including traversals, filters, and sorting. Once a SPARQL query is executed, users can visualize the results as data tables and store specific outcomes in variables to iteratively refine queries. Query results can be downloaded as JSON or CSV files, while the query itself can be saved as a TXT file.

KGSnap! implements SELECT queries on KGs using a functional SPARQL endpoint. By default, the tool is configured to query Wikidata⁴. However, users can effortlessly introduce any SPARQL endpoint of interest using a dedicated block, i.e., define a new endpoint which requires the name that will be accessible in the from clause in the select query and the endpoint URL. The interface of KGSnap! is depicted in Figure 1. As an extension of Snap!, it seamlessly integrates into the existing Snap! interface, providing users with a dedicated tab to access all KG query functionalities. While blocks from `Motion` to `Variables` belong to the Snap! platform, all blocks related to KGs are encapsulated within the `KGQueries` container. Blocks within a container share the same color, facilitating easy identification of the category to which a block belongs. Users can distinguish the container from which a block has been extracted by examining its color in relation to the category name.

In a block-based programming environment, the shape of a block determines its compatibility with other blocks and how they can be combined.

- Blocks with rounded corners are not naturally stackable due to their asynchronous nature, as seen in the SELECT query or the entity resolution process, performed via search blocks. To incorporate them into the code puzzle, they must be wrapped within a function and made synchronous, potentially by introducing a short delay between the functionality call and the return of the result(s). An example in this direction has been discussed with experts in Snap! and is visible on the bottom of Figure 1, where functions wrap search blocks and make them synchronous.
- Other blocks in KGSnap! feature a “mouth” that can enclose other blocks, enabling a hierarchical structure. For instance, the subject block can wrap one or more instances of the predicate-object block to model all predicate-object pairs sharing the same subject, as illustrated in Figure 1. Blocks can be connected via jigsaw-like connectors, facilitating their interoperability. For example, the subject block can be stacked with the filter block since they possess a compatible interface.
- Most of the proposed blocks in KGSnap! have a round shape, which facilitates their usage in completing filters. For example, the block `...@...` allows users to look up the label specified before the “@” symbol and the language specified after it (see the left side of Figure 1). Similarly, the `rdfs:label` block models the widely used predicate for

³KGSnap! source code: <https://github.com/isislab-unisa/KnowledgeGraphsAndSnap>

KGSnap! interface: <https://isislab-unisa.github.io/Snap/snap.html>

⁴Wikidata: <https://www.wikidata.org>

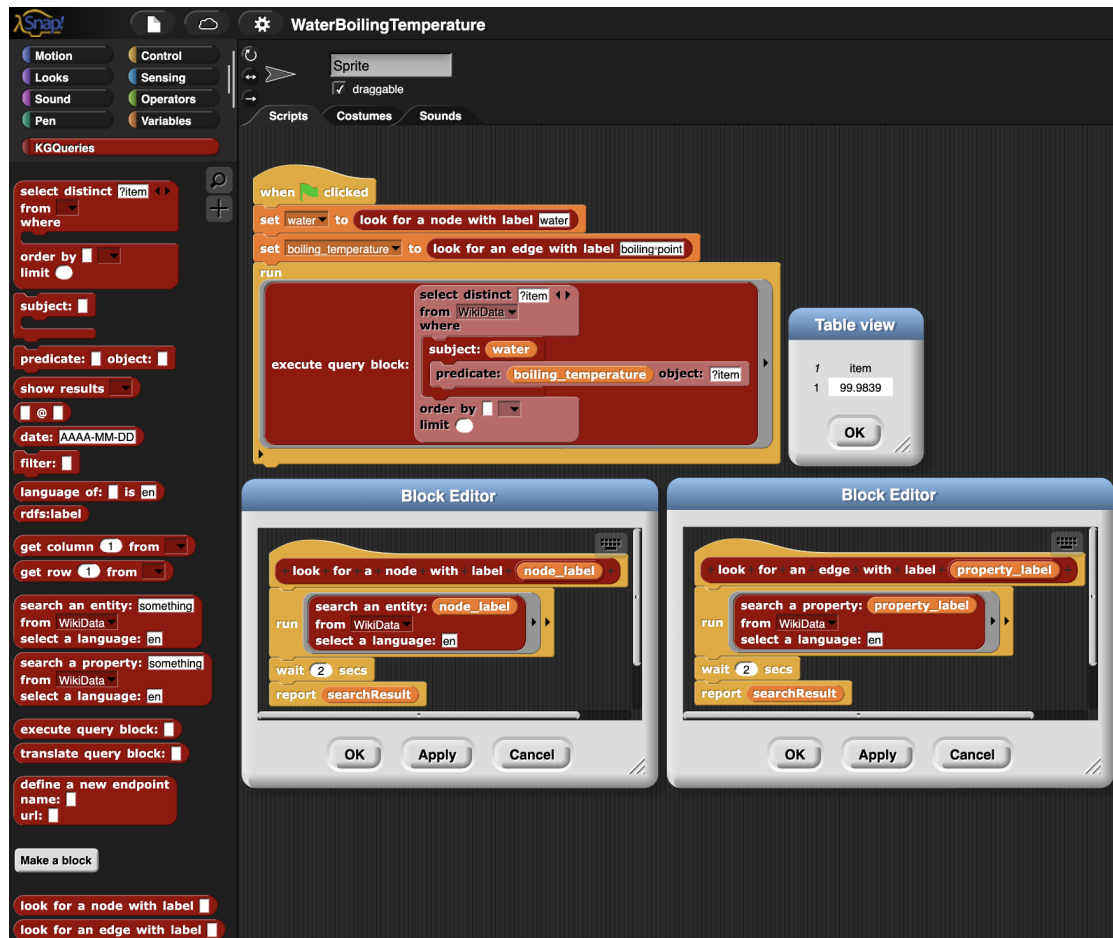


Figure 1: The KGSnap! interface. As a step-by-step tutorial, first, we define two auxiliary functions to make the search blocks synchronous. Functions are defined via the block editor and result in two monitor blocks. Then, these new blocks can be combined with the `set variable` block, as shown in the query to retrieve the boiling temperature of the water.

- attaching labels to nodes. Additionally, the `language of ... is <en>` block checks if the language of a given textual field is set to English (<en>).
- Blocks designed for authoring SPARQL SELECT queries can be enclosed within the `translate query` block to explore the formulated query. Once the query is ready, users can use the `execute query` block to run it over the configured SPARQL endpoint.
 - In addition to blocks for querying a working SPARQL endpoint, KGSnap! is equipped with blocks specifically designed for returning or manipulating results. These include the `show results` block, which displays the query results, and the `get column/row ... from` blocks, which allow users to extract specific columns or rows from the result set.

4. Ease of Use Evaluation

This section documents the evaluation of the interface of KGSnap! with end-users who lack an understanding of SPARQL. A similar evaluation protocol was repeated with two distinct groups across two separate events. KGSnap! was introduced in a dedicated workshop at Snap!Con 2023, an annual conference centered around the Snap! platform. Subsequently, KGSnap! has been tested by high-school learners during extra-curricular sessions focused on OD. While results concerning the high-school experience are documented in [31], the following will focus on the Snap!Con experience. We document participants and settings, the implemented protocol, methods of data collection, and an overview of results to assess the ease of use of KGSnap!.

Participants and Settings. The workshop was conducted online in a remote setting. Its proposal successfully passed an evaluation stage where a conference committee deemed it aligned with the Snap!Con standards and expectations. Fourteen participants freely joined the workshop without compensation. Most of them kept their cameras switched on for the entire workshop duration.

Protocol. The workshop lasted one hour and took place as an individual activity due to the remote setting. The protocol is detailed as follows:

- **Familiarization.** The moderator begins by introducing the concepts of LOD and KGs through a traditional frontal lecture supported by slides publicly available in a GitHub repository to ensure transparency and reproducibility. Then, emphasis is put on the nature of LOD, which consists of (inter)linked data published with an open license and structured in terms of nodes and edges. To illustrate these concepts, a small KG related to cultural heritage is used as an example, simulating the interest in retrieving the author of the Mona Lisa. Recognizing that SPARQL may pose challenges for learners due to its technical nature, the moderator finally introduces KGSnap! with the intention of masking these technical complexities. Once the terminology is clarified, the moderator welcomes questions from the audience, encouraging further discussion and clarification.
- **Step-by-step tutorial.** This stage is moderated as an interactive frontal lecture, challenging participants to reflect on how to respond to specific questions by hypothesizing the underlying structure of the KG. The moderator prompted participants to consider the question, *What is the boiling point of water?* Participants are encouraged to formulate hypotheses about the underlying KG and answer questions posed by the moderator to maintain engagement. Once all the participants reached a consensus on the underlying structure, the moderator formulated the query within the KGSnap! interface, as in Fig. 1.
- **Hands-on.** After 15 minutes of presentations, participants had access to the web interface of KGSnap! and could explore it autonomously. The moderator encouraged them to respond to some questions via Wikidata to engage them, but they were mostly free to interact with the interface as they preferred. The moderator was available throughout the workshop to answer questions and provide support with the platform.

Data Collection Mechanism. Participants had the option to interact with the moderator by speaking aloud or chatting with them. Additionally, the moderator provided access to their email to facilitate follow-up feedback. The results discussed in the following verbatim report all feedback spontaneously raised by participants and sent to the moderator via chat and email.

Results. Some participants autonomously executed queries and spontaneously shared their results with the moderator. For example, one participant had fun retrieving UNICODE characters via Wikidata and successfully obtained the Dutch flag. On multiple occasions, participants expressed their enjoyment and interest, stating that “It is [an activity] really fun to do and very interesting”. One participant mentioned that they attempted to query KGs directly using SPARQL two years before but failed. However, they found it much easier to perform basic queries via KGSnap! and considered it a promising access point for non-expert users due to its masking and simplification of SPARQL syntax. Other participants echoed this sentiment, saying, “It seems to me a powerful approach to use Snap! to learn about KGs”. Another participant inquired about the possibility of querying other SPARQL endpoints, allowing the moderator to introduce KGSnap! blocks for querying arbitrary working SPARQL endpoints.

Other participants primarily focused on investigating how KGSnap! blocks can be integrated with the rest of the environment to make it easier to use for the community. For instance, since KGSnap! blocks are not natively stackable as most of them perform asynchronous calls, one participant created and shared a function to make those calls synchronous, thereby enabling the possibility to fit blocks together seamlessly. Furthermore, the same participant proposed wrapping SPARQL variables in Snap! variable blocks to avoid errors that might occur when hand-writing variable names, as well as the risk of forgetting the question mark required in SPARQL to identify variables. Subsequent discussions revolved around making blocks for searching entities and properties compatible with variable blocks, as visible in Figure 1.

All participants successfully experienced KGSnap! by formulating simple but functional queries without any support. However, one participant acknowledged that KGSnap! required knowledge of entity and property names to query Wikidata, as the conceptualization stage is not masked by the interface. Despite this, the participant expressed optimism that users can easily become accustomed to it with practice. This observation can be partially confirmed by the absence of negative comments raised during the workshop.

Discussion. Similar to SparqlBlocks [26], KGSnap! provide end-users with intuitive access to commonly used structures while minimizing the need for text input. These design principles guided the interface development, where SPARQL language elements are visually represented through blocks, whose shape helps to prevent syntax errors. KGSnap! offers basic constructs that can be combined to author complex queries, along with support for more advanced features such as search class and predicate blocks, and offering commonly used classes and properties directly, such as `rdfs:label`. Moreover, query results are directly reusable, as suggested by Ceriani and Bottoni [26]. To achieve this, KGSnap! integrates results into the query view, presenting them as data tables composed of reusable variables. Results are displayed in popup windows and can be manipulated using dedicated blocks in the `KGQueries` module, such as `get columns/rows from`, which supports learners in developing their data literacy skills for

both linked and tabular data. Besides results, also queries can be exported and explored outside of the block-programming environment, as in SparqlBlocks [26].

As demonstrated by the KGSnap! ease of use evaluation, participants successfully ran simple yet functional queries on Wikidata within limited time frame, spanning from one to two hours. Moreover, KGSnap! was generally perceived as sufficiently user-friendly. It replies to our RQ.

At its current stage, KGSnap! employs a visual, action-unaware query processing approach by offering users distinct blocks for query authoring and execution, thereby precluding live query execution. Although SparqlBlocks discouraged this practice [26], none of the participants expressed concerns about this practice, likely due to their brief exposure to progressively challenging exercises. This aspect should be carefully verified in the future.

All participants remained highly engaged throughout the entire learning experiences, expressing their interest in an innovative approach to querying data that is often perceived as inaccessible by the target communities. These findings are confirmed by high-school learners [31] and echoed the strengths previously documented by the existing literature [26].

While many studies testing tools with end-users detail their protocol to enable reproducibility, educational materials supporting inexperienced educators are often not freely available. For this reason, we advocate for a broader release of educational materials as OD to promote transparency and reproducibility and foster a culture of data literacy.

5. Conclusions and Future Directions

This article explores a block-based programming approach for executing SPARQL queries without the need for technical expertise. It introduces KGSnap!, an extension of Snap! equipped with blocks designed for executing SELECT queries through basic graph patterns.

While lay users expressed satisfaction with the support provided during the learning experience, enhancing in-line support could further facilitate query formulation and block distinction within KGSnap!. Although this article evaluates the ease of use among non-experts in semantic web technologies, assessing the platform's usability with semantic web experts would be valuable to gauge their agreement with the block shapes and the implemented mechanisms for query formulation and execution. Currently, both KGSnap! and its competitors only query a single data source at a time. To enhance functionality, future efforts could focus on supporting federated queries. Encouraged by the positive feedback documented in this article, more quantitative evidence are required to verify the benefits of the proposed tool, such as the complexity of queries in terms of links followed, the time to build the queries, and a comparison with similar tools in terms of expressiveness, accuracy, and ease of use.

References

- [1] Y. Fettach, M. Ghogho, B. Benatallah, Knowledge Graphs in Education and Employability: A Survey on Applications and Techniques, *IEEE Access* 10 (2022) 80174–80183. doi:10.1109/ACCESS.2022.3194063.
- [2] C. K. Pereira, S. W. M. Siqueira, B. P. Nunes, S. Dietze, Linked Data in Education: A Survey

- and a Synthesis of Actual Research and Future Challenges, *IEEE Transactions on Learning Technologies* 11 (2018) 400–412. doi:10.1109/TLT.2017.2787659.
- [3] R. De Donato, M. Garofalo, D. Malandrino, M. A. Pellegrino, A. Petta, Education meets knowledge graphs for the knowledge management, in: *MIS4TEL, Workshops.*, Springer, 2021, pp. 272–280. doi:10.1007/978-3-030-52287-2_28.
- [4] S. Evenstein Sigalov, R. Nachmias, Investigating the potential of the semantic web for education: Exploring Wikidata as a learning platform, *Education and Information Technologies* 28 (2023) 12565–12614. doi:10.1007/s10639-023-11664-1.
- [5] B. Inostroza, R. Cid, A. Hogan, RDF Playground: An Online Tool for Learning about the Semantic Web, in: *The Web Conference*, ACM, 2023, pp. 111–114. doi:10.1145/3543873.3587325.
- [6] L. Pieschel, S. Welten, L. C. Gleim, S. Decker, Teaching Semantic Web Technologies through Interactive Jupyter Notebooks, in: *SEMANTiCS (Posters & Demos)*, 2021. URL: <https://ceur-ws.org/Vol-2941/paper6.pdf>.
- [7] M. A. Pellegrino, A. Antelmi, At school of Open Data: A literature review, in: *CSEdu, SCITEPRESS*, 2023, pp. 172–183. doi:10.5220/0011747500003470.
- [8] D. Bau, J. Gray, C. Kelleher, J. Sheldon, F. Turbak, Learnable programming: Blocks and beyond, *ACM Communication* 60 (2017) 72–80. doi:10.1145/3015455.
- [9] H. Vargas, C. B. Aranda, A. Hogan, C. López, RDF Explorer: A Visual SPARQL Query Builder, in: *ISWC*, Springer, 2019, pp. 647–663. doi:10.1007/978-3-030-30793-6_37.
- [10] P. Bellini, P. Nesi, A. Venturi, Linked open graph: Browsing multiple sparql entry points to build your own lod views, *Journal of Visual Languages & Computing* 25 (2014) 703 – 716. doi:10.1016/j.jvlc.2014.10.003.
- [11] D. Damljanovic, M. Agatonovic, H. Cunningham, Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup Through the User Interaction, in: *The Semantic Web: Research and Application*, 2010, pp. 106–120. doi:10.1007/978-3-642-13486-9_8.
- [12] V. Offertucci, M. A. Pellegrino, V. Scarano, KGSnap!: query Knowledge Graphs by Snap!, in: *The Semantic Web: ESWC Satellite Events*, Springer, 2024.
- [13] F. Haag, S. Lohmann, S. Bold, T. Ertl, Visual SPARQL querying based on extended filter/flow graphs, in: *Advanced Visual Interfaces (AVI)*, 2014, pp. 305–312. doi:10.1145/2598153.2598185.
- [14] F. Haag, S. Lohmann, S. Siek, T. Ertl, QueryVOWL: A visual query notation for Linked Data, in: *The Semantic Web: ESWC Satellite Events*, Springer, 2015, pp. 387–402. doi:10.1007/978-3-319-25639-9_51.
- [15] S. Ferré, Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language, *Semantic Web* 8 (2017) 405–418. doi:10.3233/SW-150208.
- [16] A. Soyulu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie, I. Horrocks, OptiqueVQS: A visual query system over ontologies for industry, *Semantic Web* 9 (2018) 627–660. doi:10.3233/SW-180293.
- [17] C. Yang, X. Wang, Q. Xu, W. Li, Sparqlvis: An interactive visualization tool for knowledge graphs, in: *Web and Big Data: Second International Joint Conference, APWeb-WAIM*, Springer, 2018, pp. 471–474.
- [18] S. Malyshev, M. Kröttsch, L. González, J. Gonsior, A. Bielefeldt, Getting the Most Out

- of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph, in: ISWC, Springer, 2018, pp. 376–394. doi:10.1007/978-3-030-00668-6_23.
- [19] M. A. Pellegrino, V. Scarano, C. Spagnuolo, Move cultural heritage knowledge graphs in everyone's pocket, *Semantic Web 14 (2023)* 323–359. doi:10.3233/SW-223117.
- [20] R. De Donato, M. Garofalo, D. Malandrino, M. A. Pellegrino, A. Petta, V. Scarano, QueDI: From Knowledge Graph Querying to Data Visualization, in: SEMANTiCS, Springer, 2020, pp. 70–86. doi:10.1007/978-3-030-59833-4_5.
- [21] A. Jares, J. Klimek, Simplod: Simple SPARQL Query Builder for Rapid Export of Linked Open Data in the Form of CSV Files, in: *Information Integration and Web Intelligence*, ACM, 2021, pp. 415–418. doi:10.1145/3487664.3487790.
- [22] O. Mussa, O. F. Rana, B. Goossens, P. O. ter Wengel, C. Perera, ForestQB: An adaptive query builder to support wildlife research, in: *ISWC Posters, Demos and Industry Tracks*, volume 3254, CEUR-WS.org, 2022. URL: <https://ceur-ws.org/Vol-3254/xpreface.pdf>.
- [23] P. Liu, X. Wang, Q. Fu, Y. Yang, Y.-F. Li, Q. Zhang, KGVQL: A knowledge graph visual query language with bidirectional transformations, *Knowledge-Based Systems 250 (2022)* 108870.
- [24] T. Francart, Sparnatural: A visual knowledge graph exploration tool, in: *ESWC. Satellite Events*, Springer, 2023, pp. 11–15. doi:10.1007/978-3-031-43458-7_2.
- [25] D. Le Phuoc, M. Dao-Tran, A. Le Tuan, M. N. Duc, M. Hauswirth, RDF stream processing with CQELS framework for real-time analysis, in: *International Conference on Distributed Event-Based Systems*, ACM, 2015, p. 285–292. doi:10.1145/2675743.2772586.
- [26] M. Ceriani, P. Bottoni, SparqlBlocks: Using Blocks to Design Structured Linked Data Queries, *Language (XSD) 1 (2017)*. doi:10.18293/VLSS2017-006.
- [27] A. C. Junior, C. Debruyne, D. O'Sullivan, Juma Uplift: Using a Block Metaphor for Representing Uplift Mappings, in: *International Conference on Semantic Computing (ICSC)*, IEEE, 2018, pp. 211–218. doi:10.1109/ICSC.2018.00037.
- [28] Ö. Öztürk, T. Özacar, A case study for block-based linked data generation: Recipes as jigsaw puzzles, *Journal of Information Science 46 (2020)* 419–433. doi:10.1177/0165551519849518.
- [29] A. Sanctorum, J. Riggio, S. Sepehri, E. Arnesdotter, T. Vanhaecke, O. De Troyer, A Jigsaw-Based End-User Tool for the Development of Ontology-Based Knowledge Bases, in: *International Symposium on End User Development*, Springer, 2021, pp. 169–184. doi:10.1007/978-3-030-79840-6_11.
- [30] E. W. Patton, W. Van Woensel, O. Seneviratne, G. Loseto, F. Scioscia, L. Kagal, The Punya Platform: Building Mobile Research Apps with Linked Data and Semantic Features, in: *ISWC*, Springer, 2021, pp. 563–579. doi:10.1007/978-3-030-88361-4_33.
- [31] A. Antelmi, P. Esposito, Empowering Data Literacy Among High School Learners, in: *MIS4TEL. Workshops*, Springer, 2024.