Qanary Builder: Addressing the Reproducibility Crisis in Question Answering over Knowledge Graphs

Aleksandr Perevalov[1,2,*], Andreas Both[1,3], Florian Gudat[1], Paul Bräuning[1], Johannes Meesters[1], Lennart Gründel[1], Marie-Susann Bachmann[1] and Salem Zin Iden Naser[1]

[1]*Leipzig University of Applied Sciences, Karl-Liebknecht-Straße 132, 04277 Leipzig, Germany*
[2]*DICE Group, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany*
[3]*Technology Innovation Unit, DATEV eG, Nuremberg, Germany*

Abstract

This paper discusses the challenge of reproducibility in the field of Question Answering over Knowledge Graphs (KGQA). To address this challenge, the Qanary Builder has been developed as a tool to facilitate the creation and evaluation of component-based KGQA systems. The Qanary Builder is a full-stack Web application that enables a no-code development process of KGQA systems by configuring them from pre-defined components and providing evaluation functionality. Based on the Qanary Framework, it provides visual insights and instant explainability of a KGQA process through semantic annotations. The authors aim to present the effectiveness of the Qanary Builder in addressing the reproducibility crisis and demonstrate how this tool can improve the KGQA system development and evaluation efficiency.

Keywords

Qanary Builder, Qanary Framework, Question Answering, Evaluation, Reproducibility

1. Introduction

The field of question answering over knowledge graphs (KGQA) has been increasingly important in recent years, with many applications in various domains. However, the reproducibility crisis in this field has posed a significant challenge to researchers, making it difficult to compare the effectiveness of different KGQA systems [1] and their components that represent the subtasks required for computing the answer to a given question (e.g., Named Entity Recognition). In response to this challenge, the *Qanary Builder* has been developed as a tool to facilitate the creation of *component-based KGQA systems.* The Qanary Builder extends the eponymous *Qanary Framework* [2, 3] by providing a no-code development interface, visual insights, evaluation management, and instant explainability of a KGQA process through *semantic annotations* (SAs)[1] automatically produced by KGQA components and representing the result of the corresponding sub-task of the actual KGQA process. The Qanary Framework is a core engine for processing

✉ alpe@mail.uni-paderborn.de (A. Perevalov); andreas.both@htwk-leipzig.de (A. Both);
florian.gudat.1@stud.htwk-leipzig.de (F. Gudat); paul.braeuning@stud.htwk-leipzig.de (P. Bräuning);
johannes.meesters@stud.htwk-leipzig.de (J. Meesters); lennart.gruendel@stud.htwk-leipzig.de (L. Gründel);
marie-susann.bachmann@stud.htwk-leipzig.de (M. Bachmann); salem.zin_iden_naser@stud.htwk-leipzig.de
(S. Z. I. Naser)
🌐 https://perevalov.com (A. Perevalov); http://www.andreasboth.de/en (A. Both)

[1]The SAs are stored as RDF using the Web Annotation Data Model (cf. https://www.w3.org/TR/annotation-model/).

questions. It acts as an orchestrator of different pre-defined components that can be combined in a KGQA system. The SAs are used to persist the outputs of all components of a Qanary-based system, hence, each KGQA process can be traced by following the SAs. In this regard, the KGQA systems and their components may store their confidence score, execution time, identified resources, and other information in SAs. In a nutshell, the Qanary Builder provides its users—researchers—with a full-cycle development process of KGQA systems by interactively (re-)configuring them from pre-defined components and providing built-in evaluation functionality without writing code. In this demo paper, we present the aforementioned features of the Qanary Builder and describe how it is addressing the reproducibility crisis and enabling more efficient and reliable research in this field.

2. Related Work

Reproducibility is a general challenge in many research communities. In particular, for the KGQA field, a researcher may not be able to reproduce results presented a few years ago or even the most recent ones [1]. Therefore, a number of various solutions were proposed to address this problem. The authors of [2, 3] introduce Qanary as a knowledge-based methodology for orchestrating component-based KGQA systems distributed over the Web. It employs its own RDF ontology (based on the Web Annotation Data Model) as an exchange format (Semantic Annotations) for components to build KGQA systems in a more flexible and standardized way. GERBIL [4] has been introduced as an evaluation framework for semantic entity annotation and KGQA (cf., GERBIL-QA [5]). This framework generates data in a machine-readable format and provides persistent URIs for each experiment, ensuring the reproducibility and archiving of the corresponding evaluation results. Furthermore, there were several initiatives to provide standardized benchmarks [6] and leaderboards [1] for different KGQA tasks.

3. Qanary Builder's Use Cases

The use cases that demonstrate the effectiveness of Qanary Builder in addressing the reproducibility crisis are: (1) Researchers may create their own KGQA system from available Qanary components and evaluate it on a provided dataset; (2) Researchers may take existing KGQA systems, which are represented as a single Qanary component, run the evaluation, and compare the obtained results. The use cases do not require any coding as everything is pre-defined, therefore, it standardizes the evaluation process and decreases the chances of making mistakes in between. For a better understanding, we provide a video[2] that covers Qanary Builder's use-cases and encourage readers to test the application online[3].

Figure 1 presents *the designer* module of Qanary Builder. The designer enables users to manage the available Qanary-based systems and the corresponding *configurations*, i.e., a sequential order of components to form the process of a KGQA system. The workspace of the designer allows a user to select components, try single questions to test the functionality, and see the answer as well as the SAs created by each component during the KGQA process. Thus, the designer contributes to both first and second use cases. The instant explainability is provided
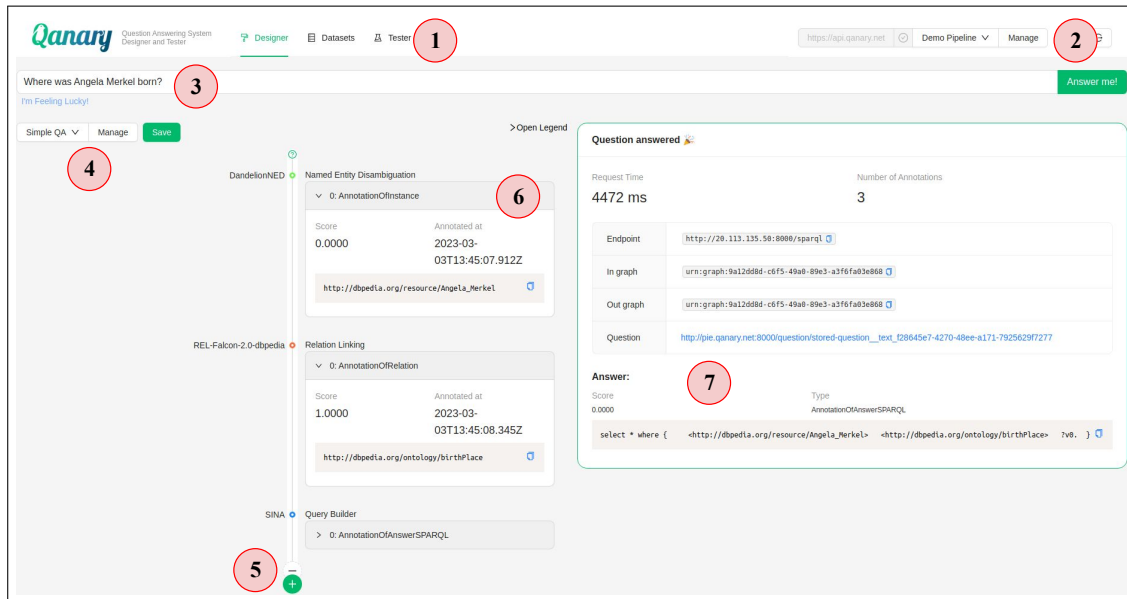
Figure 1: The designer. (1) – Menu to switch between the designer, datasets, and tester; (2) – Qanary System configuration manager; (3) – Input field for a test question; (4) – Component configuration manager; (5) – Component picker; (6) – SAs viewer (the one for the SINA component is collapsed); (7) – Answer viewer (the answer is the very last SA of a pipeline, here – the SA by the SINA component).

through the SAs viewer (Element 6 of Figure 1) to enable user directly observe what a particular KGQA component has identified. *The datasets' manager* is responsible for managing custom datasets that are further used for the evaluation. The accepted data format is a `.csv` file that contains two fields: "question" and "answer". An "answer" may be represented in different forms: a textual answer, a SPARQL query, a named entity's URI and many more. Hence, the datasets' manager is a crucial component for establishing a reproducible evaluation process related the first and second use cases. *The tester* facilitates the evaluation runs given a specified configuration and a dataset. Each run contains information on the run time, configuration, dataset, and a *question-wise accuracy score.* The tester utilizes a dataset created with the datasets' manager and iteratively sends questions to a KGQA system configuration defined in the designer. The results appear after a particular question has been processed. Therefore, the tester addresses both the first and second use cases as well.

4. Qanary Builder's Technical Overview

The Qanary Builder is split into front-end and back-end subsystems. It connects to a specified Qanary KGQA system instance[4] and monitors currently registered components. Hence, Qanary Builder always has up-to-date information on what KGQA components can be used for configuring a system. A configured KGQA system can be directly evaluated in the Qanary Builder by selecting a specific test dataset. In its turn, the test datasets are custom and are managed by a dedicated module. The overview of the architecture of Qanary Builder is presented in Figure 2.

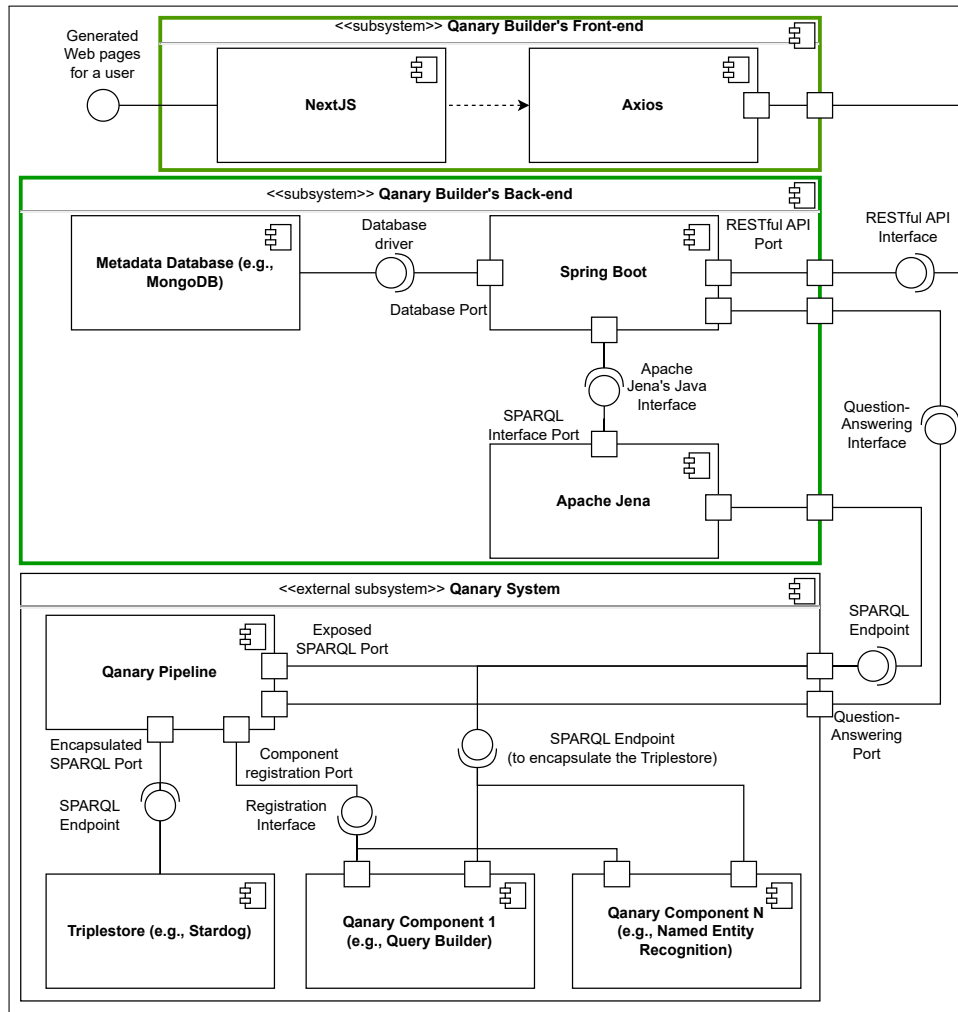[4]The Qanary was developed outside of this work.

Figure 2: The component diagram of the Qanary Builder which is connected to one of the Qanary Systems (marked as «external subsystem»). The subsystems outlined in green were developed within this paper and represent the Qanary Builder.

The front-end subsystem of the Qanary Builder is a Web application written with Next.js. It contains three functional modules: designer, datasets manager, and tester that were described in the above section. Thus, it helps users with managing their KGQA system configurations, datasets, and test runs. *The back-end* subsystem is a RESTful API written using the Spring Boot framework. It handles the logic for managing the metadata about Qanary Systems, KGQA system configurations, datasets, and test runs. The storage of this metadata is done with MongoDB via the corresponding database driver. The back-end requests the Qanary System via its Question-Answering interface to trigger processing of a question given a set of components. The back-end communicates with the Qanary System's SPARQL endpoint via Apache Jena library, which provides Java interface from one side and connects to the SPARQL endpoint from the other side. This is used to fetch the SAs and present them at the front-end subsystem.

5. Conclusion

In conclusion, the reproducibility crisis in the KGQA field has been a major concern for researchers. The development of the Qanary Builder offers a solution to this challenge by allowing the creation and evaluation of component-based KGQA systems without the need for coding. With built-in development and evaluation functionality, the Qanary Builder provides visual insights and instant explainability. By utilizing this tool, researchers can improve the reproducibility of KGQA system development and evaluation, leading to more efficient and reliable research in the field of KGQA. The source code of the whole project is published online[5] as open source (MIT License).

Acknowledgments

References

[1] A. Perevalov, X. Yan, L. Kovriguina, L. Jiang, A. Both, R. Usbeck, Knowledge graph question answering leaderboard: A community resource to prevent a replication crisis, in: Proceedings of the Thirteenth Language Resources and Evaluation Conf., 2022, pp. 2998–3007.

[2] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, C. Lange, Qanary–a methodology for vocabulary-driven open question answering systems, in: European Semantic Web Conference, Springer, 2016, pp. 625–641.

[3] D. Diefenbach, K. Singh, A. Both, D. Cherix, C. Lange, S. Auer, The Qanary ecosystem: getting new insights by composing question answering pipelines, in: ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings 17, Springer, 2017, pp. 171–189.

[4] R. Usbeck, M. Röder, A.-C. Ngonga Ngomo, C. Baron, A. Both, M. Brümmer, D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann, P. Ferragina, C. Lemke, A. Moro, R. Navigli, F. Piccinno, G. Rizzo, H. Sack, R. Speck, R. Troncy, J. Waitelonis, L. Wesemann, GERBIL: General entity annotator benchmarking framework, in: Proceedings of the 24th International Conference on World Wide Web, WWW '15, 2015, p. 1133–1143.

[5] R. Verborgh, R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A.-C. Ngonga-Ngomo, C. Demmler, C. Unger, A.-C. Ngonga Ngomo, I. Fundulaki, A. Krithara, Benchmarking question answering systems, Semantic Web 10 (2019) 293–304.

[6] R. Usbeck, R. H. Gusmita, A. N. Ngomo, M. Saleem, 9th challenge on question answering over linked data (QALD-9), in: Joint proc. of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4 and QALD-9 co-located with ISWC 2018, 2018, pp. 58–64.

---

[5] https://gitlab.imn.htwk-leipzig.de/qanary
[6] https://www.itzbund.de/