Datenbank-Spektrum (2024) 24:43–52
https://doi.org/10.1007/s13222-023-00463-0

FACHBEITRAG

Managing Provenance Data in Knowledge Graph Management Platforms

Erik Kleinsteuber[1] · Tarek Al Mustafa[1] · Franziska Zander[1,2] · Birgitta König-Ries[1,2,3] · Samira Babalou[1,2]

Received: 1 June 2023 / Accepted: 25 November 2023 / Published online: 5 February 2024

Abstract
Knowledge Graphs (KGs) present factual information about domains of interest. They are used in a wide variety of applications and in different domains, serving as powerful backbones for organizing and extracting knowledge from complex data. In both industry and academia, a variety of platforms have been proposed for managing Knowledge Graphs. To use the full potential of KGs within these platforms, it is essential to have proper provenance management to understand where certain information in a KG stems from. This plays an important role in increasing trust and supporting open science principles. It enables reproducibility and updatability of KGs. In this paper, we propose a framework for provenance management of KG generation within a web portal. We present how our framework captures, stores, and retrieves provenance information. Our provenance representation is aligned with the standardized W3C Provenance Ontology. Through our framework, we can rerun the KG generation process over the same or different source data. With this, we support four applications: reproducibility, altered rerun, undo operation, and provenance retrieval. In summary, our framework aligns with the core principles of open science. By promoting transparency and reproducibility, it enhances the reliability and trustworthiness of research outcomes.

Keywords Semantic Web · Knowledge Graph · Knowledge Graph Platform · Provenance Tracking · Reproducibility

1 Introduction

Knowledge Graphs (KGs) [1] consist of nodes, representing real-world entities, and edges connecting them, representing relationships among entities. They present factual information about a particular domain or set of domains. Nowadays, KGs are generated for different domains such as biodiversity [2], or biomedicine [3], and in various applications such as recommendation systems [4]. Both in industry and academia, different KG management platforms (cf. [5, 6]) have been suggested. A KG platform is a portal for creating, managing, and using KGs. Such platforms aim to provide functionalities and services to cover the life cycle of KGs. However, to use the full potential of KGs in any application and domain, it is essential to have proper provenance management, such that it is possible to understand where certain information in the KG stems from. This increases trust in the information provided and supports open science principles [7]. Moreover, a provenance management infrastructure in KG management platforms can ease the successful reproducibility of KGs. Reproducibility is an important concept that recreates a KG with the goal of producing the same graph and ensuring the expected results. Also, provenance management can help in recreating a graph while taking new settings into account. This helps in updating KGs effectively. All of the above are important factors in increasing the usage of KGs.

Assume a user aims to build a KG from their source data. To generate the KG, different steps are required such as cleaning, converting data, linking to other resources, or augmenting with new knowledge. The user uses multiple software tools in a specific order to generate the KG. To

Erik Kleinsteuber and Tarek Al Mustafa contributed equally to this work.

Birgitta König-Ries
birgitta.koenig-ries@uni-jena.de

✉ Samira Babalou
samira.babalou@uni-jena.de

[1] Institute for Computer Science, Friedrich Schiller University Jena, Jena, Germany

[2] German Center for Integrative Biodiversity Research (iDiv), Halle-Jena-Leipzig, Germany

[3] Michael-Stifel-Center for Data-Driven and Simulation Science, Friedrich Schiller University Jena, Jena, Germany

make the most of the KG, one may need to know: under which processes a KG is built; by whom and on which source data it is created; whether the resulting KG can be recreated with the same source data and the same processes; whether the same processes can be applied on other source data, and so on. To handle these issues, the web portal needs to include a powerful provenance management framework. Upon that, the workflow of generating a KG can be saved, rerun, edited, and applied to other source data, supporting reproducibility.

For computational tasks, it is often desired to have a record representing what happened during their execution. Generally, such records are known as provenance data. Capturing, storing, accessing, and sharing provenance data are complex problems. Solutions to these problems vary widely depending on factors like the computational environment, methods, desired provenance granularity, and much more. One solution is to integrate a computational task into an environment capable of capturing the provenance of the results of these tasks. Perez et al. [8] have systematically reviewed 25 popular provenance systems. Even though the term provenance system is not explicitly defined, it can be derived from their survey that these include, e.g., Scalable Workflow Management Systems (SWfMS) [9] or Database Management Systems (DBMS). In the survey, a taxonomy is built from the observed approaches in provenance systems that deal with different provenance issues. Moreover, Perez et al. summarized different aspects of provenance systems and discussed in detail how multiple systems addressed these aspects. They considered that the decision on a provenance solution depends on the real interests, needs, and expectations of the developer or potential users and depends heavily on the application domain.

In contrast to developing a desktop application for a single user, in web development there are usually more dependencies, restrictions, and security issues that need to be considered. In a web portal, there is often a frontend (client side), serving as a presentation layer and a possibility for the user to interact with the portal. Such interactions issue requests to the backend. The backend needs a high fault tolerance and automated functionalities to handle user requests. Extending backend functionalities with third-party tools needs security, safe execution, and careful monitoring. Such tools may come with different programming languages and dependencies. It is also essential to implement these tools such that they do not affect each other in particular or general dependency issues, writing files, or memory usage. In most web portals, developing a backend architecture with high flexibility is necessary for further implementations. Moreover, requirements on the provenance solution for the web portal might change during ongoing development.

When thinking about including a provenance platform in our studied portal, we faced a number of practical issues: Existing provenance systems (cf. the reviewed systems in [8]) tend to have constraints on programming languages and domains. It is a complex task to apply changes to big code bases when a specific feature is missing or needs to be changed. Depending on the tool's complexity, integrating third-party tools into a computational notebook can be difficult. Additionally, connecting such a provenance system or computational notebook to the web portal's functionalities adds an extra layer of complexity if possible at all.

To address the mentioned problems, in this paper, we contribute to creating a customized provenance framework in a KG generation portal with the aim of benefiting as much as possible from the collected provenance data. As a whole, our contributions can be summarized as follows:

- We introduce a framework capable of capturing the coarse-grained workflow provenance of generated KGs within a web portal. This framework is customized in our studied platform, but the underlying principle can be adapted for other platforms. We refer to the provenance data of the KG generation process as a workflow. It holds information about all executed tools and necessary inputs and outputs during KG generation.
- We ensure that the data about the computational tasks required to create the KG are stored in a reusable workflow associated with the KG and are efficiently accessible. We describe how we align captured provenance to W3C PROV-DM [10] and show the ability to store this data in RDF triple format according to W3C PROV Ontology [11].
- We associate the provenance data with two distinct approaches, operating at both the graph and triple level.
- Our provenance data can be used in four applications: reproducing a KG, undoing operations during KG generation, update a KG with adapted workflow or data, retrieve provenance info.

This idea has been initially proposed in [12]. In this revised version of our paper, we introduce several enhancements to our initial publication. Firstly, we extend the description of our proposed framework to provide a more comprehensive understanding. This allows for a deeper exploration of its potential applications. Additionally, we have extended the information covered in our literature review. We also provide a detailed explanation of the provenance data, mapped to the PROV standard, ensuring a clear and precise representation. Our new contribution here, which brings novelty to our work, is the implementation of an alternative approach for provenance storage and association in the form of triples, that also allows for provenance retrieval using SPARQL queries on the KG.

The rest of the paper is organized as follows. Sect. 2 presents the literature review. Sect. 3 shows our proposed framework, followed by implementation details in Sect. 4. We conclude the paper in Sect. 5.

2 Literature Review

The provenance of an object is the history of its origin and derivation [13]. Provenance tracking records the provenance of an object. In the literature, there have been different surveys (cf. [8, 14]) on provenance characteristics and provenance models.

Provenance data can be partitioned into two types, prospective and retrospective provenance. According to a survey by Freire et al. [15] 'Prospective provenance captures the specification of a computational task (i.e., a workflow)—it corresponds to the steps that need to be followed (or a recipe) to generate a data product or class of data products. Retrospective provenance captures the steps that were executed as well as information about the execution environment used to derive a specific data product — a detailed log of the execution of a computational task.' In this work, we focus on capturing retrospective provenance. Provenance can be captured at varying levels of detail. In this work, we collect coarse-grained provenance [16] by documenting the workflow of generating KGs from source data in a web portal.

Another important issue is finding a proper data model to represent provenance data. There have been efforts to unite the most common provenance data points into standardized specifications [17–19]. Key for our work is the W3C PROV family of specifications for modeling provenance data [20]. In detail, we chose to use 'The PROV Data Model (PROV-DM)' [10] and 'The Provenance Ontology (PROV-O)' [11]. PROV-DM [21] is a 'generic data model for provenance that allows domain and application specific representations of provenance to be translated into such a data model and interchanged between systems'. This data model uses entities, activities and agents to describe provenance. Any digital or physical thing or object, for example a dataset, may become an entity. An activity is 'something that occurs over a period of time and acts upon or with entities', e.g. an operation on said dataset. An agent is 'something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity'. Researchers generating a KG in our web portal become agents, but also external tools used to complete certain tasks can become specific types of agents, i.e. 'SoftwareAgents'. In practice, these three concepts can be connected through relationships to explain the workflow. For example, an activity can perform an action on a dataset to generate an output dataset; in PROV terms this means that the activity `prov:used` an input dataset, and that the output dataset `prov:wasGeneratedBy` the activity; and `prov:wasDerivedFrom` the input dataset. This example is visualized in Fig. 1. Additionally, the three classes also have optional attributes. Through these, it becomes possible to attach important provenance information. For example, an entity describing the input dataset can have attributes containing the original filename, type, the date of creation, or when it was uploaded to the application.

PROV-O [22] provides an encoding of PROV-DM in OWL2 Web Ontology Language [23, 24]. It provides 'a set of classes, properties, and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts'. We chose to use the PROV data model because an existing Python package (*prov* [25]) allows for straightforward implementation.

The importance of provenance on large-scale KGs and the Web of Data has been highlighted in [26]. As a solution to manage provenance, computational Notebooks (cf. Jupyter Notebook [27]) have gained widespread adoption in recent years, cf. ProvBook [28]. However, implementing large, complex projects in a notebook, especially when multiple programming languages are used, is not straightforward.

Another issue is the automation of a notebook. We faced different problems to connect a piece of software imple-

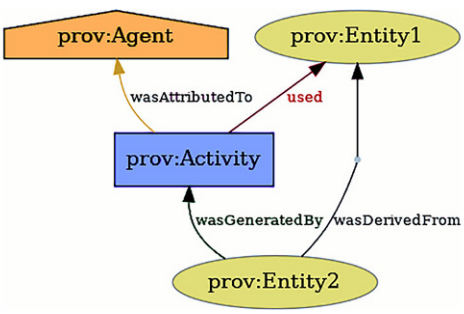

Fig. 1 Simple Provenance Example with two Entities, an Activity, and an Agent

mented in Jupyter Notebook to the backend of our studied platform, and executing cells when we receive specific user requests. On the other hand, web-based interactive development environments such as JupyterLab that can be hosted and accessed by multiple people would introduce security issues.

Existing tools such as Open Refine [29] and others can track operations applied on the data and thus capture provenance information. However, we did not use such tools as a provenance solution, as they are not a fully-fledged development environment and it is not always possible to extend those tools with arbitrary code and still make use of its provenance features.

To the best of our knowledge, a few KG platforms [5, 6, 30, 31] apply a provenance solution in some capacity. Of those, Blue Brain Nexus [6] is the only one explicitly mentioning the importance of provenance data and their usage of the W3C PROV ontology [11]. The other platforms did not explain their approach on provenance management in their publications.

3 Our Proposed Provenance Management Framework

Knowledge Graph (KG) generation is an ordered execution of tools in different phases, where source data is the input and a KG is the output. With tool, we mean an executable piece of software that performs some computational task such as cleaning, linking, or converting datasets. The input of a tool is a file along with a configuration (set of input parameters). The output of a tool is a new processed file. For simplicity, we assume that tools are run sequentially to generate a KG. This execution order needs to be preserved. We assume the input of a tool execution is the output of the prior tool execution. Every tool execution has a file as an input and a new file as an output. We define any such file as a data object.

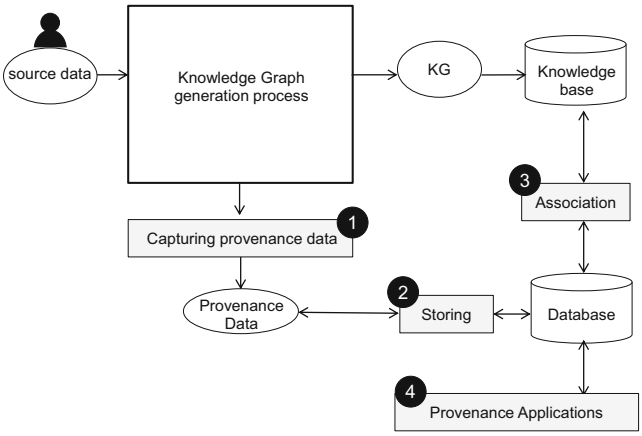

Fig. 2 Overview of our management framework for managing the provenance of the KG generation in a web portal

Fig. 2 shows our provenance management framework in the KG generation of a web portal. The users go through different phases to generate the KG based on their source data. The resulting KG is added to the knowledge base. Thus, with each generation of a new (sub-)KG, the overall KG (in the remainder of the paper referred to as main KG) is extended. The provenance data of each phase is captured (❶), stored (❷), and associated with the generated KG (❸). The provenance data can be used for various applications (❹).

3.1 Provenance Data Capture and Mapping to PROV

We capture provenance data related to the KG generation process (see ❶ in Fig. 2). Our provenance data can be mapped to the existing provenance models, namely PROV-O [11] and PROV-DM [10]. In the web portal, each KG generation starts by uploading source data by a user. After that, the user selects a tool with a specific configuration and then the tool gets executed. This happens sequentially multiple times until the KG is generated. In our framework, we capture the provenance of each data object that got processed by a tool execution during the KG generation process. All executed tools, configurations, and input and output of each phase of the KG generation are saved separately. For each, we store a set of information such as the version of the tool or the storage location of a file. All provenance data of an executed tool at every phase is captured by our system. We call the information about all executed tools in the sequential phases of the KG generation a workflow. It is an ordered collection of provenance data of executed tools in all phases of the KG generation process. Note that, the provenance data of a data object includes all stored data of prior phases until that phase. We use existing provenance vocabulary (PROV-O and PROV-DM) to structure captured data using entities, activities, and agents, following these rules:

1. A user using the web portal to generate a KG is represented as a `prov:Agent`.
2. The main KG becomes a `prov:Entity` of type `prov:Collection`.
3. Users and KG entities are connected through the `prov:wasAttributedTo` relationship to show who was responsible for a specific KG.
4. From each uploaded dataset, a sub-KG is generated. Each sub-KG is a `prov:Entity`.
5. Sub-KGs are appended to the main KG through the `prov:hadMember` relationship.

6. The workflow of our method to generate knowledge graphs becomes a `prov:Entity`, more specifically an entity of type `prov:type=prov:Plan`.
7. The provenance data of each phase of a workflow becomes a `prov:Activity`.
8. Phases are linked to the sub-KG using the `prov:wasAssociatedWith` relationship. This relationship has an additional argument to input a `prov:Plan` with which we can state that phases are executed according to the workflow.
9. The configuration of each phase (user's selections in the web portal) becomes a `prov:Entity`.
10. Phases and the configurations they use are connected using the PROV `prov:used` relationship.
11. Every tool becomes an agent, specifically a `prov:SoftwareAgent`.
12. Tools use configurations. Each configuration is saved as a `prov:Entity`.
13. Tools and their configurations are also connected using the `prov:wasAttributedTo` relationship.
14. Tools and phases are connected through the `prov:wasAssociatedWith` relationship.
15. Every phase has an input and an output data object, both mapped to `prov:Entities`. The first input data object is called the source dataset. If a data object consists of multiple parts, it becomes a `prov:Collection` and its sub-entities are append using `prov:hadMember`.
16. The phase activities and their inputs are connected through `prov:used`, while the generated outputs use the `prov:wasGeneratedBy` relationship.
17. Output and input objects connect using `prov:wasDerivedFrom`.

Fig. 3 shows an excerpt of provenance for a workflow, visualized using the *prov* Python package [25]. Blue rectangles depict activities, yellow ovals depict entities, and orange boxes describe agents. Edges depict relations according to the listing above. Note that in this example, to increase readability, prov attributes are not shown. Fig. 6 shows the RDF triples corresponding to the example in PROV-O format.

```
@prefix iknow: <https://planthub.idiv.de/iknow#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

iknow:user a prov:Agent,
       prov:Person ;

iknow:main-KG a prov:Collection,
       prov:Entity ;
    prov:hadMember iknow:sub-KG ;
    prov:wasAttributedTo iknow:user .

iknow:sub-KG a prov:Entity ;
    prov:hadPlan "iknow_workflow"^^xsd:string .

iknow:workflow a prov:Entity,
       prov:Plan .

iknow:phase_i a prov:Activity,
       prov:Collection,
    prov:qualifiedAssociation [ a prov:Association ;
           prov:agent iknow:sub-KG ;
           prov:hadPlan iknow:workflow ] ;
    prov:used iknow:config_i,
        iknow:input_i ;
    prov:wasAssociatedWith iknow:tool_i ;

iknow:input_i a prov:Entity ;
    iknow:source "dataset.csv"^^xsd:string .

iknow:output_i a prov:Collection,
       prov:Entity ;
    prov:wasDerivedFrom iknow:input_i ;
    prov:wasGeneratedBy iknow:phase_i .

iknow:tool_i a prov:Agent,
       prov:SoftwareAgent ;
    iknow:selection_tool "iknow-method"^^xsd:string .

iknow:tool_config_i a prov:Entity ;
    prov:wasAttributedTo iknow:tool_i ;
    iknow:toolconfig "iknow-method-config"^^xsd:string .

iknow:config_i a prov:Entity ;
    iknow:config "default"^^xsd:string .
```

Fig. 3 Provenance data example captured in our workflow in PROV-O syntax for a given phase i of the KG generation process; *orange boxes* show agents, *yellow ovals* present entities, and *blue rectangles* show activities; *edges* show relationships between objects as described in Sect. 3.1

3.2 Provenance data storage and KG association

During KG generation, the provenance data of each produced data object is stored (see ❷ in Fig. 2). This information ensures transparency and enables users to assess the reliability of the generated KG. Each generated KG is a sub-KG of the main KG in the web portal. The challenge lies in how we store and associate captured provenance data to its corresponding sub-KG (see ❸ in Fig. 2). We differentiate between graph-level and triple-level associations. Graph-level association represents how captured provenance data is connected to its corresponding KG. Triple-level association shows how a specific triple is coupled to its provenance. In our work, we consider two storage and association approaches using either the knowledge base itself, or a relational database. The second approach has been implemented in our previous work [12]. The new, first approach, is presented below.

Approach 1 – Triple-based KG Provenance Storage: In this approach, KGs and their provenance data are saved

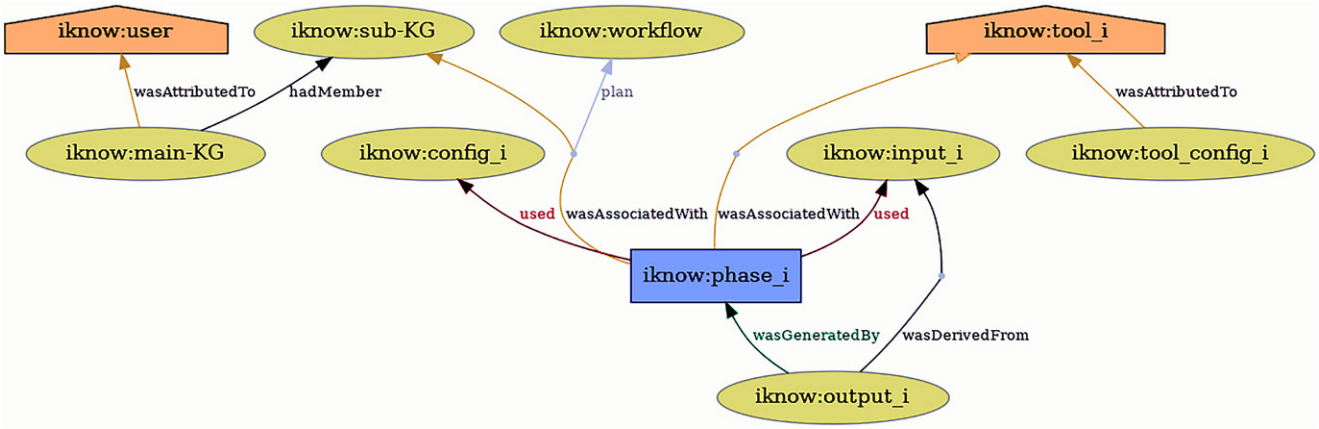

Fig. 4 Visualisation of Approach 1 with an example KG. Nodes attached to the *left* of `sub-KG` show the KG itself. Nodes on the *right*, attached to `prov-sub-KG`, contain provenance information

in the knowledge base. This requires the provenance data to be represented as triples. The aforementioned provenance capture approach allows for generating output files in different file formats, including TTL. As such, the files can be saved in the knowledge base. In the current implementation, once provenance data is captured, it is associated with the corresponding sub- and main-KG at runtime, such that there exists a triple connecting the sub-KG to the main-KG (see Fig. 6) that can be retrieved by using a query containing the sub-KGs identifier. This way, provenance is associated at graph-level. Since all provenance data for the workflow is connected to the sub- and main-KG in triple form through the `prov:wasAssociatedWith` relationship (see Fig. 3), the triple-level can be queried by providing the identifiers of the main or sub-KG and additional information about the specific parameter, e.g., which phase or tool should be retrieved.

Currently, coarse-grained provenance about the workflow is tracked. Each input dataset is associated with its own sub-KG. Therefore, triples contain information about how a dataset is transformed at each phase. However, the provenance capture approach does not have the granularity to track how each entry of the input dataset was transformed by each step.

Fig. 4 shows a visualisation of an example KG in GraphDB [32]. The example KG combines actual graph information (cell, row, and column values), attaches it to a node representing the sub-KG, couples it to a node representing the sub-KG's provenance (through `owl:sameAs`) and attaches all provenance data points to that node. Fig. 5 shows in more detail, how the provenance nodes and edges are represented in the KG. Querying the workflow provenance in the KG itself therefore becomes straightforward.

Approach 2 – RDBMS-based KG Provenance Storage: The generated KGs are stored in a knowledge base, while their provenance data is stored in a relational database. This separation allows for efficient organization and retrieval of provenance information. To handle the provenance storage in the web portal, we save provenance data of the KG generation process (i.e., workflow) as a JSON-string inside the provenance record. For each new KG generation, a new entry in the relational database is created. This entry contains provenance data about each phase of the workflow and is called provenance record. Each KG generation process has a primary key and belongs to a specific project. Here, both graph- and triple-levels are supported. At the graph-level, the reference (URI of the sub-KG in the knowledge base) of a sub-KG is stored in the relational database. This ensures a direct link between the sub-

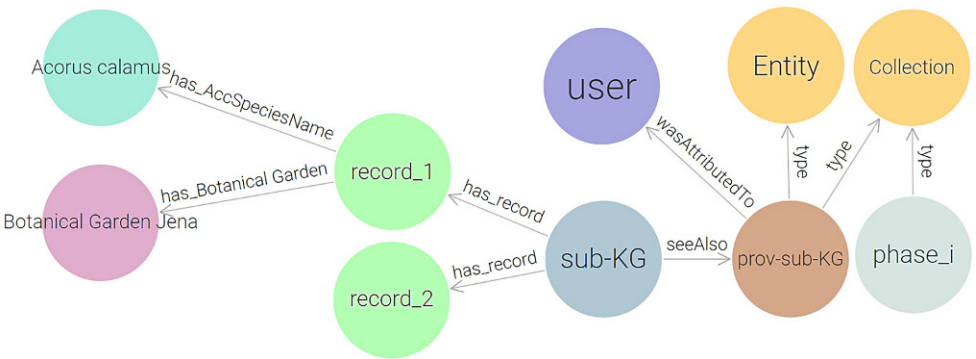

Fig. 5 Visualisation of provenance data in the KG. Shown here are data points associated with the phases of KG generation

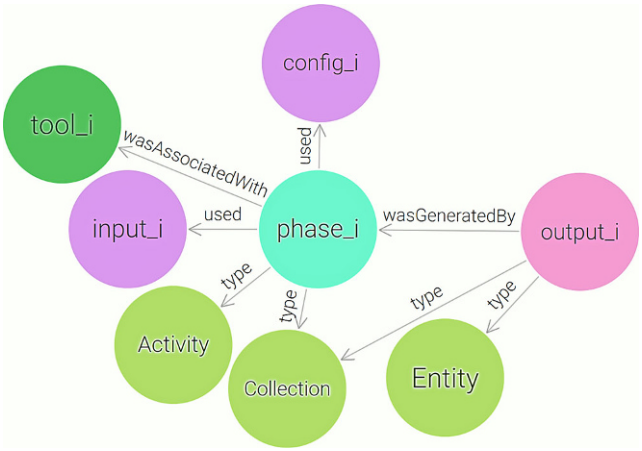

Fig. 6 Example of a workflow holding one phase in RDF

KG and its provenance data. Therefore, each sub-KG in the knowledge base has an associated reference (id) in its provenance data. At the triple-level, subjects, predicates, and objects of all triples are annotated (e.g., via `rdfs.seeAlso`, `rdfs.comment`, or other defined annotations) with their respective provenance ids.

The provenance data of a specific sub-KG or a term (subject, predicate, object) can be retrieved by a lookup via the reference of the sub-KG and optional provenance id. The association is of type "no-coupling" according to [8]. The associated provenance data can be stored in different formats such as JSON, XML, or TTL.

3.2.1 Comparison of Approaches

In this section, we provide the respective strengths and weaknesses of the two provenance storage approaches presented earlier.

Approach 1 – Advantages:

- Smooth integration with existing knowledge base structure.
- Directly aligns with the RDF-based representation of the KG.
- Provides comprehensive provenance data about the entire workflow.
- Provenance can be retrieved by querying the graph itself. No further technologies are necessary.

Approach 1 – Disadvantages:

- Limited granularity in tracking transformations of individual input dataset entries. However, in future work there may be solutions on how to include and associate fine grained provenance data. Smaller granularity might increase the graph's complexity considerably.
- By saving provenance in the KG itself, the number of triples grows with each data source used. This uses more storage space, might increase the complexity of queries, but also can hinder a persons ability to understand the KG.

Approach 2 – Advantages:

- Efficient organization of provenance data.
- Allows for structured storage and easy accessibility/retrieval of provenance records.
- Provenance is not stored in the KG itself. This mitigates disadvantages mentioned above.

Approach 2 – Disadvantages:

- More complex implementation compared to Approach 1. This complexity arises from the need to establish and manage the database schema and ensuring compatibility with the existing infrastructure. Additionally, the process of integrating and querying data from the relational database requires specialized knowledge in database management. Moreover, maintaining synchronization between the knowledge base and the relational database introduces an additional layer of complexity.

It becomes evident that each method presents distinct advantages and disadvantages. The choice between these approaches will depend on specific project requirements, resource availability, and the desired depth of provenance tracking.

4 Implementation and Application

This section gives a detailed description of the implementation of our provenance framework within the iKNOW project and its applications.

4.1 Implementation

In this section, we provide detailed insights into the technical implementation, including an overview of the technologies used.

About iKNOW: Our provenance framework is a part of the iKNOW platform [33]. iKNOW aims to create a semantic-based toolbox for Knowledge Graph creation and evolution in the biodiversity domain. Within iKNOW, we had the opportunity to run initial tests of the provenance framework using real-world data. The implementation of our framework is available under an open-source license [34].

Backend and Frontend Technologies: For the backend, we use the Python web framework Django [35] due to its

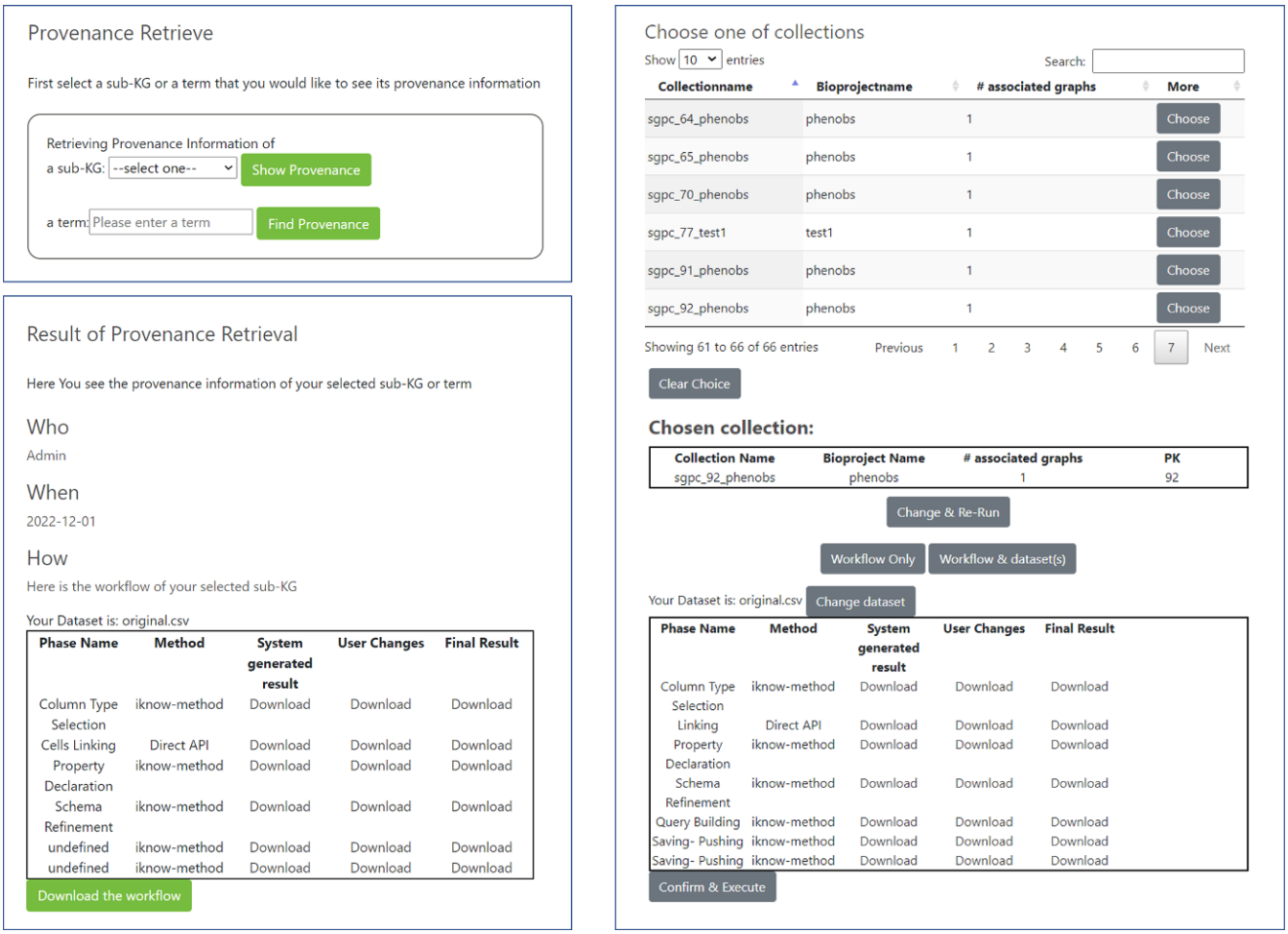

Fig. 7 *Top-Left:* Provenance retrieval GUI – Users select which term or sub-KG they want to retrieve the provenance for; *Bottom-Left:* result of provenance retrieval; *Right:* The GUI for the altered rerun scenario

flexibility in data operations. For building user interfaces on the frontend, we use Svelte [36], SvelteKit [37], and Skeleton [38] for their smooth integration and efficiency in creating dynamic and user-friendly interfaces.

Dependency Management: We use Docker [39] to encapsulate different tools. A Docker container packages up code and all associated dependencies. This prevents dependency issues and provides an isolated runtime environment.

Data Management and Storage: We use PostgreSQL [40] for managing data of the portal functionalities and provenance data of Approach 2 (see Sect. 3.2). Moreover, we use Blazegraph [41] for storing and accessing KGs.

Provenance Storage Approach: The implementation of the RDBMS-based approach of storing provenance (Sect. 3.2) has been described previously [12]. We implemented the new approach of provenance storage and association using the *prov* python package.

4.2 Applications

As a whole, our provenance framework can support four different applications (see ❹ in Fig. 2):

1 – Reproducibility. A reproducible KG increases trust in the information it contains and supports open science principles. Reproducing a KG is the process of re-executing all steps that lead to the creation of the original KG to gain the same result. Reproducing the KG can be done by re-running a pre-existing workflow of a sub-KG (captured through provenance data). The application of this process is automatic within our framework. Through the GUI, users can select an existing workflow and re-run the entire process. Afterwards, the resulting KG can be downloaded separately. Users can also compare the new KG to the original by comparing their triples and metadata shown in the web portal. One should note, though, that the same results can only be achieved with the same input. Thus, in the case of dynamic input sources that do not easily allow accessing earlier versions, reproducibility is only possible, if copies

of the sources are stored. This is not automatically provided by our framework.

2 – Altered rerun. To compare or achieve better results based on different tools or configurations used within an existing workflow, executing it again with small changes should be possible. This can also include exchanging the source data. This is particularly important, if KGs are based on non-static data sources that are updated frequently or extended.

A user might want to generate a new KG based on an existing workflow of a pre-generated KG while possibly changing tools, configurations, or even source data. To achieve this, as we have shown in Fig. 7, right users can select a workflow, apply desired changes to it, and rerun the workflow over the same or other source data. The main advantage here is the possibility of generating a new KG automatically with an already known workflow. This provides user convenience. In the end, the altered workflow is saved as a new workflow in the web portal.

3 – Undo operation. When generating a new sub-KG, executed tools may not produce the expected results. Therefore, it should be possible to undo the last operations. During the KG generation process in the iKNOW portal, users can roll back one or several executed tool(s). If this functionality is used, provenance will be updated accordingly. Some additional implementation details, e.g. deleting files or maintaining the consistency of provenance data and files in the database, must be considered to ensure the safety of the operation.

4 – Provenance retrieval. Retrieving provenance data is important for users to interpret and use data correctly. Through our GUI, users can select which sub-KG they want to retrieve the provenance data from (see Fig. 7, left). It is also possible to retrieve the provenance data of a specific term. The system first searches on which sub-KG the queried term exists. It then shows the list of sub-KGs to the user. Afterwards, users can select one of the sub-KGs to see its provenance data. Users can observe by who, when, and how the sub-KG was built. It is currently possible to download provenance data of a sub-KG as a TTL file according to the first approach, and as a JSON file according to the second approach.

5 Conclusion

We proposed a framework showing an environment capable of provenance management to capture, store and retrieve provenance data of Knowledge Graph generation in a web portal. We have presented how our provenance data can be mapped to PROV-DM and PROV-O. Also, we introduced two different approaches for its storage and retrieval. Moreover, we presented four different applications to show the benefit of our proposed framework. The framework has been implemented as part of iKNOW, a project aiming at providing a platform for KG generation for the biodiversity domain. In our future work, we will leverage this setting to perform extensive user tests with real world biodiversity data and scientists from that domain. This will help to validate or make design choices, but also provide valuable insights into the general usability and usefulness of the framework. Additionally, we plan for several possible extensions of the framework, including extending the provenance capture and storage for tools with special requirements outside of our definition. This can involve, e.g., multiple inputs and outputs of a tool.

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Nickel M, Murphy K, Tresp V, Gabrilovich E (2015) A review of relational machine learning for knowledge graphs. IEEE 104(1):11–33
2. Page RD (2019) Ozymandias: a biodiversity knowledge graph. PeerJ 7 (2019): e6739.
3. Wood E, Glen AK, Kvarfordt LG, Womack F, Acevedo L, Yoon TS, Ma C, Flores V, Sinha M, Chodpathumwan Y et al (2022) Rtx-kg2: a system for building a semantically standardized knowledge graph for translational biomedicine. BMC Bioinform 23(1):400
4. Wu J, Zhu X, Zhang C, Hu Z (2020) Event-centric tourism knowledge graph— a case study of hainan. In: KSEM. Springer, pp 3–15
5. Haase P, Herzig DM, Kozlov A, Nikolov A, Trame J (2019) metaphactory: a platform for knowledge graph management. SW 10(6):1109–1125
6. Sy MF, Roman B, Kerrien S, Mendez DM, Genet H, Wajerowicz W, Dupont M, Lavriushev I, Machon J, Pirman K et al (2023) Blue brain nexus: an open, secure, scalable system for knowledge graph management and data-driven science. Semantic Web 14.4 (2023): 697-727. https://doi.org/10.3233/SW-222974
7. Samuel S, König-Ries B (2022) End-to-end provenance representation for the understandability and reproducibility of scientific experiments using a semantic approach. J Biomed Semant 13:1
8. Pérez B, Rubio J, Sáenz-Adán C (2018) A systematic review of provenance systems. Knowl Inf Syst 57(3):495–543

9. Abdul M (2016) Scalable Scientific Workflows Management System SWFMS. International Journal of Advanced Computer Science and Applications 7.11.
10. Belhajjame K, B'Far R, Cheney J, Coppens S, Cresswell S, Gil Y, Groth P, Klyne G, Lebo T, McCusker J et al (2013) Prov-dm: The prov data model. W3c Recomm 14:15–16
11. Lebo T, Sahoo S, McGuinness D, Belhajjame K, Cheney J, Corsar D, Garijo D, Soiland-Reyes S, Zednik S, Zhao J (2013) The prov ontolog. In: Prov-o
12. Kleinsteuber E, Babalou S, König-Ries B (2023) A provenance management framework for knowledge graph generation in a web portal. In: BTW 2023
13. Majumdar R, Meyer R, Wang Z (2013) Provenance verification. In: International Workshop on Reachability Problems. Springer, pp 21–22
14. Herschel M, Diestelkämper R, Lahmar HB (2017) A survey on provenance: what for? What form? What from? VLDB J 26(6):881–906
15. Freire J, Koop D, Santos E, Silva CT (2008) Provenance for computational tasks: a survey. Comput Sci Eng 10(3):11–21
16. Braun U, Garfinkel S, Holland DA, Muniswamy-Reddy K-K, Seltzer MI (2006) Issues in automatic provenance collection. In: Provenance and Annotation of Data: International Provenance and Annotation Workshop, IPAW 2006 May 3-5, 2006 Springer, Chicago, IL, USA, pp 171–183
17. Moreau L, Freire J, Futrelle J, McGrath RE, Myers J, Paulson P (2008) The open provenance model: An overview. In: International provenance and annotation workshop. Springer, pp 323–326
18. Moreau L, Clifford B, Freire J, Futrelle J, Gil Y, Groth P, Kwasnikowska N, Miles S, Missier P, Myers J et al (2011) The open provenance model core specification (v1. 1). Future Gener Comput Syst 27(6):743–756
19. Cheney J, Finkelstein A, Ludäscher B, Vansummeren S (2012) Principles of provenance (dagstuhl seminar 12091). In: Dagstuhl Reports, vol 2. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
20. Missier P, Belhajjame K, Cheney J (2013) The w3c prov family of specifications for modelling provenance metadata. In: Proceedings of the 16th International Conference on Extending Database Technology, pp 773–776
21. Prov-dm. https://www.w3.org/TR/prov-dm/. Accsessed Feb 2023
22. Prov-o. https://www.w3.org/TR/prov-o/. Accsessed Feb 2023
23. Hitzler P, Krötzsch M, Parsia B, Patel-Schneider PF, Rudolph S et al (2009) Owl 2 web ontology language primer. W3c Recomm 27(1):123
24. Owl2 web ontology language. https://www.w3.org/TR/owl2-overview/. Accsessed Feb 2023
25. prov python package. https://pypi.org/project/prov/. Accsessed Feb 2023
26. Hogan A (2020) Web of data. In: The Web of Data. Springer, pp 15–57
27. Kluyver T, Ragan-Kelley B et al (2016) Jupyter notebooks-a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B (eds) In Positioning and Power in Academic Publishing: Players,Agents and Agendas, vol 2016. IOS Press, pp 87–90
28. Samuel S, König-Ries B (2018) Provbook: Provenance-based semantic enrichment of interactive notebooks for reproducibility. In: ISWC. P&D/Industry/BlueSky, (in)
29. Open refine. https://openrefine.org/. Accsessed Feb 2023
30. Staar PWJ, Dolfi M, Auer C (2020) Corpus processing service: a knowledge graph platform to perform deep data exploration on corpora. Appl Ai Lett 1(e20):2
31. Berven A, Christensen OA, Moldeklev S, Opdahl AL, Villanger KJ (2020) A knowledge-graph platform for newsrooms. Comput Ind 123:103321
32. Ontotext graphdb. https://www.ontotext.com/products/graphdb/. Accsessed Feb 2023
33. Babalou S, Schellenberger Costa D, Kattge J, Römermann C, König-Ries B (2021) Towards a semantic toolbox for reproducible knowledge graph generation in the biodiversity domain – how to make the most out of biodiversity data. In: INFORMATIK 2021. Gesellschaft für Informatik, Bonn, pp 581–590
34. iKNOW github repository. https://github.com/fusion-jena/iKNOW. Accsessed Feb 2023
35. Django web framework. https://www.djangoproject.com/. Accsessed Feb 2023
36. Svelte. https://svelte.dev/. Accsessed Feb 2023
37. Sveltekit. https://kit.svelte.dev/. Accsessed Feb 2023
38. Skeleton. https://www.skeleton.dev/. Accsessed Feb 2023
39. Docker. https://www.docker.com/. Accsessed Feb 2023
40. Postgresql. https://www.postgresql.org/. Accsessed Feb 2023
41. Blazegraph. https://www.blazegraph.com/. Accsessed Feb 2023