Full Traceability and Provenance for Knowledge Graphs

Henrik DIBOWSKI[a,1]
[a] *Robert Bosch GmbH, Bosch Research, Bosch Center for Artificial Intelligence, 71272 Renningen, Germany*
ORCiD ID: Henrik Dibowski https://orcid.org/0000-0002-9672-2387

Abstract. Knowledge graphs (KGs) continue spreading into industrial use cases due to their advantages and superiority over classical data representations. A problem that has not yet adequately been solved for KGs is the traceability and provenance of changes, which can be required in an enterprise or by regulations. KGs typically contain the current snapshot of data valid at a certain moment in time only. Changes over time are usually not recorded and no change history exists. The lack of suitable and scalable traceability solutions hinders the wider application of KGs. This paper presents a traceability and provenance solution for KGs, which can track all changes of a KG on triple level. It comprises a provenance engine that intercepts SPARQL/Update queries; PROV-STAR, an RDF-star enabled light-weight extension of the Provenance Ontology (PROV-O) for representing changes and their provenance; and a SPARQL query transformation approach for tracking the changes on a separate provenance KG with SPARQL-star queries. The solution supports full traceability of all changes, on the lowest possible level of triples, with each change being comprehended with detailed provenance information. From the provenance KG a detailed change history can be retrieved, and any past version of the KG can be restored with a single query. The implementation and validation have shown that changes can be tracked at runtime during the normal operation of a KG. Furthermore, the solution is scalable to large KGs and frequent updates, as only the delta of each change is stored.

Keywords. Knowledge Graph, Ontology, Traceability, Provenance, Change History, RDF-star, SPARQL-star

1. Introduction

The popularity and application of knowledge graphs (KGs) have strongly increased over the past years, with KGs spreading further into industrial use cases due to their advantages and superiority over classical data representations. KGs typically contain a snapshot of information that is valid at a certain moment in time. Driven by users, applications, algorithms, data ingestion pipelines etc. the information changes over time, which effectively means that new triples are inserted or existing triples are deleted. In RDF triple stores, information can be changed by SPARQL/Update queries. If not explicitly taken care of, no change history is available for a KG, i.e. all older states and changes over time are lost, but only the current snapshot of data remains. This prohibits the traceability of changes, provisioning of detailed provenance information of each

[1] Corresponding Author: Henrik Dibowski, henrik.dibowski@de.bosch.com.

change, undo and repair functionality, the recovery of old states, insights about the evolution of information (and things) over time and more, which are highly desirable functionalities. For many applications and use cases it is even a hard regulatory or enterprise-level requirement to have a change history available.

A few solutions and approaches exist that can record a change history and that try to overcome the above limitations, but none is capable to solve all limitations. Their data representation is often inefficient, the unnatural representation of triples is cumbersome to update and query, and/or provenance information is missing.

This paper presents a novel traceability and provenance solution for KGs, which can track all changes of a KG on triple level and supports full traceability. It comprises several main contributions: 1) a provenance engine module acting as a middleware that intercepts SPARQL/Update queries; 2) PROV-STAR, an RDF-star enabled lightweight extension of the Provenance Ontology (PROV-O) for representing changes and their provenance; and 3) a SPARQL query transformation approach for tracking the changes on a separate provenance KG with SPARQL-star queries. From the provenance KG a detailed change history can be retrieved (change history functionality), and any past version of the KG can be restored from it with a single query (recovery functionality).

The remaining paper is structured as follows: the related work is described in Section 2, followed by a description of the solution in Section 3, and its evaluation in Section 4. Section 5 concludes the paper and provides an outlook on future research directions.

2. Related Work

Traditionally, *provenance* has been used on time-stamping a digital document [1], to prove lineage of data in scientific computing [2], for scientific workflows [3], and more lately to provide metadata about the origin of data [4], [5].

Our understanding of provenance is how the W3C Provenance Working Group defines provenance, namely as the "information about entities, activities and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness" [6]. The W3C PROV specifications include the PROV data model (PROV-DM) [6] and the PROV Ontology (PROV-O) [7], an OWL2 ontology that allows the data model to be mapped to RDF, TRiG, and Turtle. For other definitions of provenance and how they evolved see [8].

Traceability refers to the degree to "which data is well documented, verifiable, and easily attributed to a source" [9]. This dimension is highly related to provenance metadata. For instance, data has metadata of the source, of the changes made, etc [10].

Expressing machine-interpretable statements in the form of subject-predicate-object triples is a well-established practice for capturing semantics of structured data. However, the standard used for representing these triples, RDF, inherently lacks the mechanism to attach provenance data, which would be crucial to make automatically generated and/or processed data authoritative [11].

One way to overcome this limitation and to define metadata with RDF is *reification*, which is also used for modeling n-ary relations [12]. It allows us to express statements about statements, for instance provenance. The RDF database GraphDB offers a "data history and versioning" plugin, which enables users to access past states of a database through versioning of the RDF data model level [13]. Although the change history is saved in a relational database table, it applies query transformation and reification to allow querying the history with SPARQL. Provenance information is not recorded.

Eccenca Corporate Memory, a commercial semantic data management software, supports the versioning of graph changes in a separate versioning graph via reification [14]. It therefore applies the changeset vocabulary, which defines a set of terms for describing changes to resource descriptions [15]. The delta between two versions of a resource is represented by two sets of triples: additions and removals. Triples therein are represented as reified RDF statements. Some of the downsides of reification is the high amount of triples needed, as it requires minimum four additional triples per reified statement [16], and the unnatural representation of triples, which is cumbersome and complex to read and write with SPARQL.

RDF document-level provenance is realized by TerminusDB: a document-oriented in-memory graph database with version control and collaboration model available under Apache 2.0 license [17]. TerminusDB allows multiple versions of a KG to be stored efficiently, with deltas containing relative additions and deletions, as with GIT. It can query past commits, but therefore requires to use GIT like commands for creating a branch, for computing the difference between branches etc. The offered granularity is low, with commits typically comprising larger sets of changes on a single file.

Named graphs have been proposed in the past to overcome the issues with RDF reification, or knowledge-based specific extensions including quads and contexts [18]. Furthermore, the lossless decomposition of RDF graphs and tracking their provenance using *RDF molecules*, which is claimed to be the finest and lossless component of an RDF graph, has been proposed [19]. The level of granularity of RDF molecules is between RDF document level and RDF triple level. An approach to create a summary graph capturing temporal evolution of entities across different versions of a KG using RDF molecules is described in [20]. A clear disadvantage of RDF molecules is the need to apply a complex decomposition algorithm that requires to query data, which does not scale for large KGs and which is not useable for runtime-tracking of provenance of a KG.

Hartig proposed an alternate approach to RDF reification called *RDF-star* (or alternatively RDF*), which embeds triples into (metadata) triples as the subject or object [21]. RDF-star is an extension of RDF's conceptual data model and concrete syntaxes, which enables the creation of concise triples that reference other triples as the subject and object resources [22]. Triples that include a triple as subject or an object are known as RDF-star triples or *quoted triple patterns*, and the included triples as *embedded triples* or *quoted triples*.

RDF-star can accomplish *triple-level provenance*, i.e. the provisioning of provenance information for RDF triples, which is adequate for a number of applications. This is the most granular provenance level for RDF data, because it can represent the provenance of statements, which is adequate for a number of applications [11]. Only little research has been done in the direction of representing metadata with RDF-star, with the most mature approach being StarVers [22]. It applies the NQT-SP temporal metadata representation model, i.e. nested quoted triples, to bundle the time of creation and time of deletion of a triple together, as can be seen in Figure 1. By doing so, for each triple the time interval when it is or was valid is explicitly given. One disadvantage of using nested quoted triples is query performance, as many RDF-star systems are not optimized for it or cannot handle arbitrary nesting levels. StarVers furthermore requires the update of existing triples and the replacement of the expiry date upon changes, which impacts the performance. Most importantly, StarVers does not support provenance.

```
<< << subject pred object >> vers:valid_from t1 >> vers:valid_until tE .
```

Figure 1. StarVers: Annotating a triple with two temporal properties using nested quoted triples.

3. Full Traceability and Provenance Solution

This paper presents a solution that realizes a full traceability of all changes on a KG in the lowest possible granularity, i.e. on triple level, and their provenance. The overall solution and workflow is shown in Figure 2 and explained in the following. It comprises several main contributions:

1. *Provenance engine*: a software module and middleware that realizes the traceability functionality. It intercepts and transforms incoming SPARQL/Update queries.
2. *PROV-STAR Ontology*: an RDF-star enabled lightweight extension of the PROV Ontology for representing changes and provenance on triple level. See Section 3.1.
3. *Provenance knowledge graph*: Applies the PROV-STAR Ontology and represents the entire changes on the original KG with RDF-star. See Section 3.2.
4. *Query transformation approach*: An approach for transforming SPARQL/Update queries into provenance-enabled SPARQL-star/Update queries for tracking the changes on the provenance KG. See Section 3.3.
5. *Change history functionality*: retrieving and filtering the change history with SPARQL-star. See Section 3.4.
6. *Recovery functionality*: restoring any old version of the original KG with SPARQL-star. See Section 3.5.

The core idea is to save a history of all changes on a KG, called the "*original knowledge graph*" (original KG), in a separate graph, called the "*provenance knowledge graph"* (provenance KG). The provenance KG is a mirror of the original KG enriched with the entire history of changes and rich provenance information. It can track each individual update operation on the KG on triple-level.

The solution applies RDF-star [22] for stating provenance information on each added and deleted triple and hence accomplishes the lowest possible granularity. The paper introduces the *PROV-STAR Ontology*, a lightweight extension of PROV-O [7] by a few new classes and an RDF-star enabled property. By that enhancement of PROV-O, the W3C standard can now be used for representing provenance on triple-level.

At the core of the solution shown in Figure 2 is the *provenance engine*, which is a middleware between the user and a SPARQL engine. All update operations, i.e. all SPARQL/Update queries, to be executed on the original KG are intercepted and transformed by the provenance engine. Therefore, the provenance engine realizes the following steps:

1. It intercepts all incoming SPARQL/Update queries.
2. It forwards the queries to the original KG, without applying changes.
3. It transforms queries with the *query transformation approach* and executes the provenance-enabled SPARQL-star/Update queries on the provenance KG.

The provenance engine thus makes sure that all changes on the original KG are completely tracked live and at runtime on the provenance KG, either in a separate dataset, or in a named graph of the original KG. The rationale for mirroring and enriching all information in the provenance KG, instead of enriching the original KG itself, is to make the provenance tracking an optional feature, and to assure the same performance of operations on the original KG, without negative impact on the querying performance or available storage capacity.

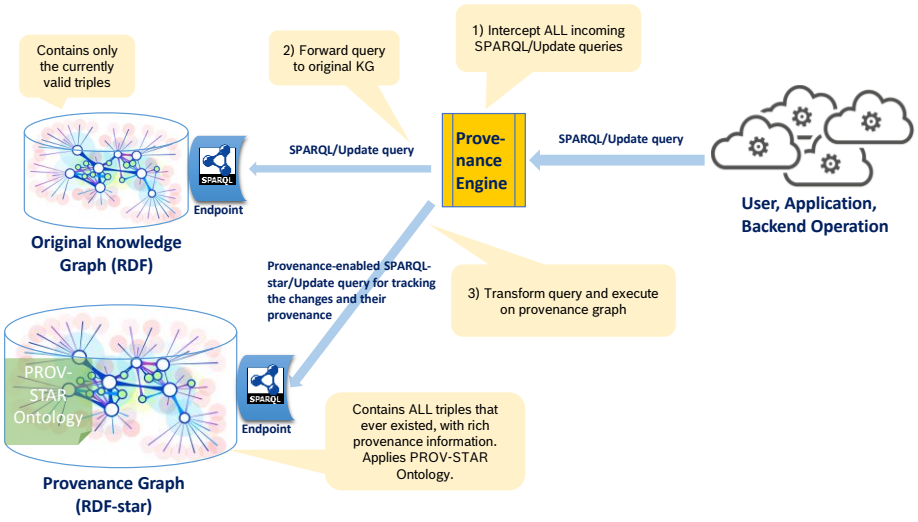

Figure 2. Full traceability of knowledge graphs – Solution overview and workflow.

By the chosen representation of the provenance KG with RDF-star and the PROV-STAR Ontology, all past information can be queried natively with SPARQL-star. With a SPARQL-star query the change history can be retrieved and filtered (*change history functionality*, see Section 3.4), or any past version of the original KG can be recovered (*recovery functionality*, see Section 3.5)

In the following sections, the different main contributions are explained in detail.

3.1. RDF-Star Traceability and Provenance Ontology

PROV-O [7] is a W3C standardized provenance vocabulary for representing provenance information in RDF. As pointed out in Section 2, RDF is not directly capable of describing metadata for triples. Hence, when using RDF, the PROV-O can natively represent provenance information for the nodes of a KG, but expressing provenance for property values requires workarounds such as reification, or RDF-star. The traceability solution applies and extends PROV-O with the capability to represent provenance information for RDF triples, the atomic pieces of information in a KG and hence the lowest possible granularity. The PROV-STAR Ontology introduced in this paper therefore extends PROV-O by three new classes and one new property, as can be seen in Figure 3 as UML class diagram. Figure 9 (see Appendix) reveals the source code of the PROV-STAR Ontology in Turtle.

With the PROV-STAR Ontology we intend to be compliant with PROV-O to the highest possible extent and use the available concepts whenever possible. The main classes `prov:Entity`, `prov:Activity` and `prov:Agent`, along with the object properties `prov:wasGeneratedBy`, `prov:wasInvalidatedBy`, `prov:wasAssociatedWith` and the datatype properties `prov:generatedAtTime` and `prov:invalidatedAtTime` of PROV-O remain the core vocabulary for representing provenance information. The PROV-STAR Ontology refines the class `prov:Entity` by three subclasses:

- `provs:TripleChangeSet`: Direct subclass of `prov:Entity`. Constitutes a class for representing sets of triples that were changed in an update operation.
- `provs:TripleGenerationSet`: A subclass of `provs:TripleChangeSet`. For representing sets of triples that were inserted.
- `provs:TripleInvalidationSet`: A subclass of `provs:TripleChangeSet`, and sibling class of `provs:TripleGenerationSet`. For representing sets of triples that were deleted

The new object property `provs:belongsTo` is RDF-star enabled, meaning that it can be used for attaching provenance facts, i.e. instances of `provs:TripleGenerationSet` and `provs:TripleInvalidationSet`, to the added and deleted triples. The triples herein appear as embedded triples in the subject position.

Another extension of PROV-O is the class `provs:Query` (see Figure 3), which optionally allows for attaching the SPARQL/Update query that caused the change as a string to the activity in order to save it along with the changed triples, if needed.

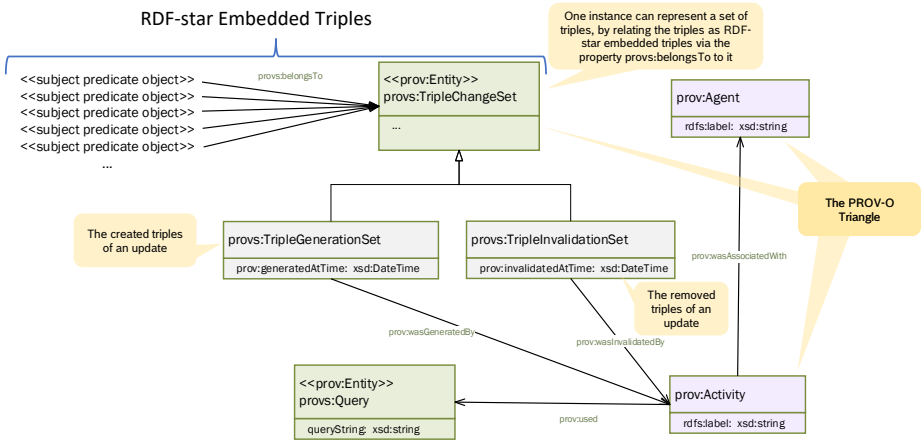

Figure 3. The PROV-STAR Ontology: an RDF-star enabled lightweight extension of the PROV Ontology. Prefix "prov": PROV Ontology, prefix "provs": PROV-STAR Ontology.

3.2. RDF-star Representation of Changes in the Provenance Knowledge Graph

The atomic pieces of information in a KG are RDF triples, comprising a subject, predicate and object. This is the smallest granularity for representing information, and on which level changes can happen. The PROV-STAR Ontology (see Section 3.1) provides the vocabulary for representing changes on this smallest granularity of triples.

Let's take a look at how changes on a KG are represented in RDF-star with the PROV-STAR Ontology. In Figure 4 a SPARQL/Update query is given comprising a DELETE DATA and an INSERT DATA subquery. The query changes the 'is released' status of some appliance control unit instance from 'false' to 'true' and adds its 'release date'. The DELETE DATA subquery therefore deletes one triple, namely the value of the `o:isReleased` datatype property, and the INSERT DATA subquery asserts the new value for it, and the value for the `o:releaseDate` datatype property.

The effects of running the query on the original KG are represented with the PROV-STAR Ontology in RDF-star in the provenance KG, as shown in Figure 5. The deleted

triple appears as RDF-star embedded triple in the subject position of another triple. It associates it via the object property `provs:belongsTo` with a new `provs:TripleInvalidationSet` instance. The two inserted triples are associated as embedded triples with a new `provs:TripleGenerationSet` instance. The remaining triples define details of the `provs:TripleInvalidationSet` and `provs:TripleGenerationSet` instances, such as the time of the change, as well as a `prov:Activity` instance representing the activity that has caused the change, and a `prov:Agent` instance representing the person that has triggered the change.

With this representation, a precise chronological change history of all triples that were deleted and inserted in a KG over time, along with provenance information comprising the time of change, activity and agent, can be saved on the provenance KG. All changes that ever happen can be recorded and traced back.

```
DELETE DATA {
  d:ACU_0b9aff9c o:isReleased  false .
};
INSERT DATA {
  d:ACU_0b9aff9c o:isReleased  true .
  d:ACU_0b9aff9c o:releaseDate "2023-11-12"^^xsd:date .
}
```

Figure 4. SPARQL/Update example query. It changes the 'is released' flag of an appliance control unit instance from 'false' to 'true' and adds a 'release date' value.

```
<< d:ACU_0b9aff9c o:isReleased  false >>
  provs:belongsTo s:TripleInvalidationSet_2834417e .
<< d:ACU_0b9aff9c o:isReleased  true >>
  provs:belongsTo s:TripleGenerationSet_2834417e .
<< d:ACU_0b9aff9c o:releaseDate "2023-11-12"^^xsd:date >>
  provs:belongsTo s:TripleGenerationSet_2834417e .
s:TripleInvalidationSet_2834417e
  rdf:type provs:TripleInvalidationSet ;
  prov:invalidatedAtTime "2023-11-13T12:36:18.444Z"^^xsd:dateTime ;
  prov:wasInvalidatedBy s:Activity_2834417e;
.
s:TripleGenerationSet_2834417e
  rdf:type provs:TripleGenerationSet ;
  prov:generatedAtTime "2023-11-13T12:36:18.444Z"^^xsd:dateTime ;
  prov:wasGeneratedBy s:Activity_2834417e ;
.
s:Activity_2834417e
  rdf:type prov:Activity ;
  rdfs:label "released ACU and added a release date" ;
  prov:wasAssociatedWith s:Agent_Peter_Kampl ;
.
s:Agent_Peter_Kampl
  rdf:type prov:Agent ;
  rdfs:label "Peter Kampl" ;
.
```

Figure 5. RDF-star representation of the effect of the query from Figure 4, stored in the provenance knowledge graph.

3.3. Query Transformation Approach for Change Tracking

For saving the history of all changes using the RDF-star representation introduced in the previous section, a *query transformation approach* is applied by the provenance engine. An incoming SPARQL/Update query is transformed into a provenance-enabled SPARQL-star/Update query, which can record the change in the provenance KG.

The query transformation is based on a predefined SPARQL-star INSERT DATA query template, into which the updated triples as well as provenance information need to be inserted. This is straightforward and can be implemented in a programming language of choice. The SPARQL-star INSERT DATA query template, already customized for the query from Figure 4, is presented in Figure 6. When executed on the provenance KG, this query generates the triples shown in Figure 5.

```
INSERT DATA {
  << d:ACU_0b9aff9c o:isReleased  false >>
    provs:belongsTo s:TripleInvalidationSet_2834417e.
  << d:ACU_0b9aff9c o:isReleased  true >>
    provs:belongsTo s:TripleGenerationSet_2834417e.
  << d:ACU_0b9aff9c o:releaseDate "2023-11-12"^^xsd:date >>
    provs:belongsTo s:TripleGenerationSet_2834417e.
  s:TripleInvalidationSet_2834417e rdf:type provs:TripleInvalidationSet.
  s:TripleInvalidationSet_2834417e prov:invalidatedAtTime
    "2023-11-13T12:36:18.444Z"^^xsd:dateTime.
  s:TripleInvalidationSet_2834417e prov:wasInvalidatedBy
    s:Activity_2834417e.
  s:TripleGenerationSet_2834417e rdf:type provs:TripleGenerationSet.
  s:TripleGenerationSet_2834417e prov:generatedAtTime
    "2023-11-13T12:36:18.444Z"^^xsd:dateTime.
  s:TripleGenerationSet_2834417e prov:wasGeneratedBy
    s:Activity_2834417e.
  s:Activity_2834417e rdf:type prov:Activity.
  s:Activity_2834417e rdfs:label
    "released ACU and added a release date".
  s:Activity_2834417e prov:wasAssociatedWith s:Agent_Peter_Kampl.
  s:Agent_Peter_Kampl rdf:type prov:Agent.
  s:Agent_Peter_Kampl rdfs:label "Peter Kampl".
}
```

Figure 6. Provenance-enabled SPARQL-star INSERT DATA query (template) for saving a change on the provenance knowledge graph, on the example of the query from Figure 4. The grey and yellow highlighted information contain prefilled example data, see Figure 5.

The query transformation requires to insert the deleted and inserted triples into the first lines (grey highlighted information) as embedded triples, each having a `provs:belongsTo` relationship to one and the same `provs:TripleInvalidationSet` and `provs:TripleGenerationSet` instance. The IRIs of the two instances, and all other used instances, need to be unique IRIs, which can be accomplished by attaching a unique string to the end of their instance identifiers, here "`2834417e`". The following lines define the details of the two instances, i.e. their `rdf:type`, the time of change (here the time when the original KG got updated is to be inserted), and a reference to an activity instance. The activity itself comes with a label that describes the change, and it refers to the agent it was triggered by. The name of the agent, either a person or an application, is to be provided as string. When executed on the provenance KG, this query generates the triples shown in Figure 5.

The generated INSERT DATA queries are very efficient as they simply push data without querying or updating any existing data. The queries can be executed independently and asynchronously on the provenance KG, i.e. it is not required to follow the chronological order of the changes on the original KG. The time of change stated in the query ensures that no matter in what sequence the queries are executed, the chronological order is preserved in the change history.

3.4. Change History of a Knowledge Graph

With the described solution, all changes applied to a KG can be saved in the provenance KG, down to each individual triple. This valuable information can be queried with SPARQL-star and displayed, for instance in a UI. The SPARQL-star query shown in Figure 7 can just do that: it provides all changes that happened after a given moment in time, chronologically sorted along the time of change, starting with the latest change. It returns the time of change, the action (i.e. deletion of a triple, or insertion of a triple), the subject, predicate and object of the changed triple, and provenance information, i.e. the name of the agent responsible for the change, and a description of the action. This can be seen in Table 1 for the example data given in Figure 5.

Table 1. Change history example, retrieved with the SPARQL-star query from Figure 7.

time	action	Subject	pred	object	agentLabel
2023-11-13T12:36:18.444Z	CREATED	d:ACU_0b9aff9c	o:isReleased	true	Peter Kampl
2023-11-13T12:36:18.444Z	CREATED	d:ACU_0b9aff9c	o:releaseDate	2023-11-12	Peter Kampl
2023-11-13T12:36:18.444Z	DELETED	d:ACU_0b9aff9c	o:isReleased	false	Peter Kampl

```
SELECT ?time ?action ?subject ?pred ?object ?agentLabel ?activityLabel
WHERE {
  {
    ?tripleChangeSet rdf:type provs:TripleGenerationSet.
    ?tripleChangeSet prov:wasGeneratedBy ?activity.
    ?tripleChangeSet prov:generatedAtTime ?time.
    FILTER (?time > "2023-11-13T12:00:00.000Z"^^xsd:dateTime)
    BIND ("CREATED" AS ?action).
  } UNION {
    ?tripleChangeSet rdf:type provs:TripleInvalidationSet.
    ?tripleChangeSet prov:wasInvalidatedBy ?activity.
    ?tripleChangeSet prov:invalidatedAtTime ?time.
    FILTER (?time > "2023-11-13T12:00:00.000Z"^^xsd:dateTime)
    BIND ("DELETED" AS ?action).
  }
       ?activity rdfs:label ?activityLabel.
       ?activity prov:wasAssociatedWith ?agent.
       ?agent rdfs:label ?agentLabel.
       <<?subject ?pred ?object>> provs:belongsTo ?tripleChangeSet.
}
ORDER BY DESC(?time) ?action ?subject ?pred ?object
```

Figure 7. SPARQL-star query for fetching the history of changes that happened on a KG after a given time, starting from the latest change. Can be enhanced with additional filtering, e.g. agent, activity, time interval.

3.5. Recovery of Past Versions of a Knowledge Graph

Besides querying the change history of a KG, the change tracking approach supports an even more powerful functionality: the recovery of any past version of a KG. Since all changes that ever happened are completely traced on the provenance KG, all information is available for restoring any past version from there. This can be easily accomplished by executing the SPARQL-star query given in Figure 8 on the provenance KG.

The query returns a graph that contains all triples that existed at a given moment in time (variable `?pastDateTime`). It therefore iterates over all triples that were created before that time (i.e. instances of `provs:TripleGenerationSet` with `prov:generatedAtTime` values older than `?pastDateTime`), and that have not been invalidated until that time (i.e. there is no `provs:TripleInvalidationSet` instance between the time of creation and the given `?pastDateTime` which states the deletion of the triple).

The restored graph is identical with the original KG as it was at that past moment in time. It can be queried with normal SPARQL queries without any limitations.

```
CONSTRUCT { ?subject ?pred ?object }
WHERE {
  BIND( "2023-08-30T10:30:54.761Z"^^xsd:dateTime AS ?pastDateTime)
  ?tripleGenerationSet a provs:TripleGenerationSet.
  ?tripleGenerationSet prov:generatedAtTime ?timeOfGeneration.
  FILTER (?timeOfGeneration <= ?pastDateTime).
    <<?subject ?pred ?object>> provs:belongsTo ?tripleGenerationSet.
    FILTER NOT EXISTS {
      <<?subject ?pred ?object>> provs:belongsTo ?tripleInvalidationSet.
      ?tripleInvalidationSet prov:invalidatedAtTime ?timeOfInvalidation.
      FILTER (?timeOfInvalidation > ?timeOfGeneration &&
                ?timeOfInvalidation <= ?pastDateTime)
    }
}
```

Figure 8. SPARQL-star query for recovering a past version of a knowledge graph at a given time ?pastDateTime from the provenance knowledge graph.

4. Evaluation

This section evaluates the introduced traceability and provenance approach for KGs. It discusses and demonstrates the main features and reports on findings from the practical usage. The approach has been fully implemented by our software development teams and is in use within Bosch.

The main features of the approach are:

1. Full traceability of all changes on a KG
2. The change history of a KG can be saved live at runtime
3. Scalability to large KGs by storing only the delta
4. Past information and its provenance can be retrieved via queries
5. Any past version of a KG can be restored

Feature 1 ensures that all possible updates of a KG, i.e. the insertion and deletion of triples, can be stored on the provenance KG, such that all changes are fully traceable over

the entire lifetime of a KG. As explained in this paper, the approach can trace changes on the finest granularity level of triples, with the smallest possible change being just a single triple. Our implementation could validate that in all respects. This can be reproduced by anyone interested by applying the data and queries presented in the paper, which cover all steps of the traceability approach.

The remaining features 2-5 are evaluated in the following with the implemented solution running on a HPE ProLiant DL380 Gen10 server with 20 Intel Xeon Silver 4114 CPUs at 2.20GHz (40 cores in total), 188 GB RAM, 447 GB SSD and Ubuntu 20.04.6 LTS. We tested the solution with two different triple stores: 1) Stardog, version 9.2.0, commercial solution, and 2) Apache Jena Fuseki, version 4.10.0, open source.

Table 2 shows details of a provenance KG that was used for the evaluation, along with certain characteristics. It has a size of 5.15 million triples and comprises the change history of an original KG comprising 63,799 triples at the latest state. Both datasets contain data from Bosch Home Comfort, in particular semantic models of residential heating systems and heat pumps, including their components, hardware, firmware etc. The provenance KG could be restored from a GIT repository in which all the changes over the past two years were tracked, with an update interval of one hour between the individual commits. As Table 2 shows, there were 10,809 changes, with over 5 million updated triples in total. The number of updated triples per change range from 1 to 500, with an average of 475.68 updated triples.

Table 2. Provenance Knowledge Graph analyzed in the evaluation and its characteristics.

No. triples		No. changes	No. updated triples per change			No. updated triples in total
Original KG	Provenance KG		Min	Max	Average	
63,799	5,159,540	10,809	1	500	475.68	5,062,259

Feature 2 enables the use of the traceability solution live at runtime, when users or applications make changes on a KG. Then the provenance engine can record the change history on the provenance KG simultaneously. Data only needs to be pushed with an INSERT DATA query to the provenance KG (see Section 3.3), and it is not required to query or update existing data. The proposed solution is highly optimized for simultaneous changes on both KGs and beats comparable solutions, such as StarVers, which requires changes on outdated triples. We simulated concurrent changes of ten users, which could be handled by our implementation live at runtime. The query transformation typically runs in just a few milliseconds and can be neglected. The execution time of the generated INSERT DATA query on the provenance KG was measured in the range of hundred milliseconds to two seconds, as can be seen in Table 3 for two measured example queries: 1) the INSERT DATA query with three updated triples from Figure 6; 2) a similar INSERT DATA query comprising 500 deleted and 500 inserted triples. This proved sufficient for using it at runtime.

Feature 3 ensures the scalability of the solution to large KGs, as it only needs to store the delta of each change on the provenance KG. Looking at the ratio between the total number of updated triples (5,062,259), and the number of triples in the provenance KG (5,159,540), which is just 1:1.019, it can be seen that the amount of additional triples for representing the provenance information amounts to only 1.9% of the overall data, in the example. In general, the amount of the overhead depends on the granularity of the update operations, whether only a few triples are updated at once, or multiple. In the worst case,

when only one triple is updated, the ratio of triples is 1 (original KG) to 9 triples (provenance KG), which constitutes a large overhead. With three updated triples (both deleted and inserted), as in the example of Figure 5, it is 1 to 4 (three updated triples on original KG against 12 on provenance KG). With 100 updated triples, the triple overhead is rather small, with a ratio of 100 to 112. The larger the update operations are, the more efficient is the representation on the provenance KG. With each update, the size of the provenance KG grows linearly with the number of updated triples. Overall, the proposed RDF-star representation is very resource efficient. When using reification, for example, the required triple count would be at least four times higher, as the representation of each reified triple itself requires minimum four triples. As modern triple stores are capable of storing and querying even billions of triples efficiently, we presume that the scalability of our solution to large KGs comprising millions of triples is ensured.

Table 3. Query execution times on the provenance knowledge graph in milliseconds.

Query	Fuseki Dataset	Stardog Dataset
INSERT DATA query with 3 triples (see Figure 6)	0.101 s	0.152 s
INSERT DATA query with 1000 triples	0.979 s	2.110 s
Change history query (see Figure 7), on interval with 60 updated triples	0.286 s	0.425 s
Change history query, on interval with 1.9M updated triples, limit 1000	163.7 s	24.949 s
Change history query, on interval with 60 updated triples, and given subject instance	0.268 s	0.151 s
Change history query, on interval with 1.9M updated triples, and given subject instance, limit 1000	3.809 s	0.179 s
Recovery query with 48K restored triples (see Figure 8)	53.140 s	39.346 s

Feature 4 denotes the capability to query any past information from the provenance KG, along with provenance information. This was demonstrated in Section 3.4 by means of a SPARQL-star query (Figure 7). As can be seen in Table 3, the query execution times can vary significantly, from just 0.286 to 163.7 s. The more triples the update interval contains, which can be controlled via an upper and/or lower date boundary inside the query (see FILTER statement in Figure 7), the longer the query execution takes. For large provenance KGs it is advisable to always use date boundaries, as otherwise the query needs to iterate over the entire data. The query in Figure 7 proved to be much more efficient when narrowing the search space by additional filters, such as a prebound instance. This variation allows for retrieving the change log of a single instance efficiently. As Table 3 shows, even for an update interval comprising 1.9M triples, the query succeeds within a few hundred milliseconds on Stardog.

The most significant functionality of the solution is the capability to recover any past version of a KG (Feature 5), as described in Section 3.5. The dataset recovery query (Figure 8) has provided proof to restore an entire past version of a KG from the provenance KG within less than a minute, as the query execution times in Table 3 show. The validation provides a first indication about the different performance of Fuseki and Stardog. While Fuseki appears to be faster in the data population task, Stardog significantly outperforms Fuseki in all other queries, except for the first change history query. It must be noted that the RDF-star support of Stardog is in beta, until RDF-star has been standardized. Further performance improvements can be expected in the future.

5. Conclusion and Outlook

This paper presents a triple-level traceability and provenance solution for KGs. It comprises a provenance engine that intercepts SPARQL/Update queries; PROV-STAR, an RDF-star enabled lightweight extension of the provenance ontology (PROV-O) for representing the changes and their provenance natively with RDF-star; and a SPARQL query transformation approach. Our validation and implementation of the solution have shown its capability to fully trace all changes on a KG, to be scalable to large KGs by storing only the delta, to retrieve past information and its provenance via SPARQL-star queries, and to restore any past version of a KG by running a single SPARQL-star query. With the generated INSERT DATA queries not requiring any queries on or updates of existing information, the solution is highly optimized for ensuring a fast writing of the change history on the provenance KG, live at runtime of a system. The chosen RDF-star data representation is slim and superior to other representations and saves about 75% of triples compared to reification as the most widely used solution so far.

The solution provides the basis for several advanced functionalities, which can be developed on top of it. Undo, repair and merge operations can now be realized, since with each intermediate state of the data and its provenance all required information is available. From the available change history valuable insights could be derived, such as comprehensive metrics and their evolution over time. Having the temporal aspect available in the provenance KG, a multitude of questions and queries that were not supported before can be processed and answered, such as ‘how has the factory evolved over time?’, or ‘how has the complexity of systems changed in the past five years?’. Furthermore, the combination of KGs with machine learning, the so-called neuro-symbolic AI, can be lifted to a totally new dimension and complexity. Machine learning could be applied for deriving insights and predicting coming changes from the (known) evolution of things, for example.

References

1. Haber, S.; Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology (3), 99–111 (1991), https://doi.org/10.1007/BF00196791
2. Simmhan, Y.L.; Plale, B.; Gannon, D.: A survey of data provenance in e-Science. ACM Sigmod Record 34(3), 31–36 (2005), https://doi.org/10.1145/1084805.1084812
3. Davidson, S.B.; Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, pp 1345–1350, Association for Computing Machinery, New York, NY, United States (2008), https://doi.org/10.1145/1376616.1376772
4. Moreau, L.; Growth, P.; Miles, S.; Vazquez-Salceda, J. Ibbotson, J.; Jiang, S.; Munroe, S.; Rana, O.; Schreiber, A.; Tan, V. et al.: The provenance of electronic data. Communications of the ACM 51(4), 52–58 (2008), https://doi.org/10.1145/1330311.1330323
5. Seneviratne, O.W.: Data Provenance and Accountability on the Web. In: Sikos, L.F.; Seneviratne, O.W.; McGuinness, D.L. (eds) Provenance in Data Science, Advanced Information and Knowledge Processing, Springer, Cham (2021), https://doi.org/10.1007/978-3-030-67681-0_2
6. Belhajjame, K.; B'Far, R., Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P. ; Klyne, G.; Lebo, T.; McCusker, J.; Miles, S.; Myers, J.; Sahoo, S., Tilmes, C.: PROV-DM: The PROV Data Model. W3C recommendation (2013), https://www.w3.org/TR/prov-dm/

7. Belhajjame, K.; Cheney, J.; Corsar, D.; Garijo, D.; Soiland-Reyes, S.; Zednik, S.; Zhao, J.: PROV-O: The PROV Ontology. W3C recommendation (2013), https://www.w3.org/TR/prov-o/
8. Chapman, A.; Sasikant, A.; Simonelli, G.; Missier, P.; Torlone, R.: The Right (Provenance) Hammer for the Job: A Comparison of Data Provenance Instrumentation. In: Sikos, L.F.; Seneviratne, O.W.; McGuinness, D.L. (eds) Provenance in Data Science, Advanced Information and Knowledge Processing, Springer, Cham (2021), https://doi.org/10.1007/978-3-030-67681-0_3
9. Wang, R.Y.; Strong, D.M.: Beyond accuracy: what data quality means to data consumers. Journal of Management Information Systems 12(4), 5–33 (1996)
10. Fensel, D.; Simsek, U.; Angele, K.; Huaman, E.; Kärle, E.; Panasiuk, O.; Toma, I.; Umbrich, J.; Wahler, A.: Knowledge Graphs – Methodology, Tools and Selected Use Cases. Springer, Cham (2020)
11. Sikos, L.F.; Philp, D.: Provenance-Aware Knowledge Representation: A Survey of Data Models and Contextualized Knowledge Graphs. Data Science and Engineering 5, 293–316 (2020), https://doi.org/10.1007/s41019-020-00118-0
12. Hayes, P.; Welty, C.: Defining N-ary Relations on the Semantic Web. W3C Working Group Note (2006), http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/
13. Ontotext: Data history and versioning. GraphDB documentation (2023), https://graphdb.ontotext.com/documentation/10.0/data-history-and-versioning.html
14. Eccenca GmbH: Versioning of Graph Changes. Eccenca documentation (2022), https://documentation.eccenca.com/22.1/explore-and-author/versioning-of-graph-changes/
15. Tunnicliffe, S.; Davis, J.: Changeset – A vocabulary for describing changes to resource descriptions (2005), https://vocab.org/changeset/schema
16. Orlandi, F.; Graux, D.; O'Sullivan, D.: Benchmarking RDF Metadata Representations: Reification, Singleton Property and RDF. In: 15th IEEE International Conference on Semantic Computing (ICSC), pp. 233-240, Laguna Hills, CA, USA (2021), doi: 10.1109/ICSC50631.2021.00049
17. TerminusDB: An Open Source Graph Database & Document Store (2023), https://terminusdb.com/products/terminusdb/
18. Watkins, E.R.; Nicole, D.A.: Named Graphs as a Mechanism for Reasoning About Provenance. In: Zhou, X.; Li, J.; Shen, H.T.; Kitsuregawa, M.; Zhang, Y. (eds) Frontiers of WWW Research and Development - APWeb 2006, Lecture Notes in Computer Science, vol 3841. Springer, Berlin, Heidelberg (2006), https://doi.org/10.1007/11610113_99
19. Ding, L.; Finin, T.; Peng, Y.; Da Silva, P. P.; McGuinness, D. L.: Tracking rdf graph provenance using rdf molecules. In: Proc. of the 4th International Semantic Web Conference (Poster), pp. 42, Springer (2005)
20. Tasnim, M.; Collarana, D.; Graux, D.; Orlandi, F.; Vidal, M.E.: Summarizing Entity Temporal Evolution in Knowledge Graphs. In: Companion Proceedings of the 2019 World Wide Web Conference (WWW '19), pp. 961–965, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3308560.3316521
21. Hartig, O.: Foundations of RDF⋆ and SPARQL⋆ (An Alternative Approach to Statement-Level Metadata in RDF). In: Workshop on Foundations of Data Management, Montevideo, Uruguay (2017)
22. Hartig, O.; Champin, P. A.; Kellogg, G.; Seaborne, A.: RDF-star and SPARQL-star. W3C final community group report (2021), https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html
23. Kovacevic, F.; Ekaputra, F.J.; Miksa, T.; Rauber, A.: StarVers - Versioning and Timestamping RDF data by means of RDF-star - An Approach based on Annotated Triples. Semantic Web 0 (2022), 1–17 (2022), https://www.semantic-web-journal.net/content/starvers-versioning-and-timestamping-rdf-data-means-rdf-star-approach-based-annotated

Appendix

Note: This part (i.e. ontology) will be removed from the final paper and instead be provided via a GitHub repository. This will make the paper match the given page limit of 14 pages overall.

```
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix provs: <http://www.bosch.com/semantics/prov-star#> .

<http://www.bosch.com/semantics/prov-star>
  a owl:Ontology ;
  owl:imports <http://www.w3.org/ns/prov-o#> ;
  owl:versionInfo "Created by Henrik Dibowski" ;
.
provs:TripleChangeSet
  a owl:Class ;
  rdfs:comment "The set of triples changed during some graph update
      operation will be captured by the two subclasses of
      TripleChangeSet" ;
  rdfs:label "Triple Change Set" ;
  rdfs:subClassOf prov:Entity ;
.
provs:TripleGenerationSet
  a owl:Class ;
  rdfs:comment "Represents a set of triples created during some graph
      update operation. They will refer to the same TripleGenerationSet
      via statement-level annotations (RDF*)." ;
  rdfs:label "Triple Generation Set" ;
  rdfs:subClassOf provs:TripleChangeSet ;
.
provs:TripleInvalidationSet
  a owl:Class ;
  rdfs:comment "Represents a set of triples deleted during some graph
      update operation. They will refer to the same TripleInvalidation
      Set via statement-level annotations (RDF*)." ;
  rdfs:label "Triple Invalidation Set" ;
  rdfs:subClassOf provs:TripleChangeSet ;
.
provs:belongsTo
  a owl:AnnotationProperty ;
  rdfs:comment "for relating triples as quoted triples to
      TripleGenerationSet and TripleInvalidationSet instances
      (statement-level annotation using RDF*)" ;
  rdfs:label "belongs to" ;
  rdfs:range provs:TripleChangeSet ;
```

Figure 9. The PROV-STAR Ontology in turtle RDF syntax.