Tracking Changes in RDF(S) Repositories

Damyan Ognyanov, Atanas Kiryakov

OntoText Lab, Sirma AI EOOD, 38A Chr. Botev blvd, 1000 Sofia, Bulgaria
{damyan, naso}@sirma.bg

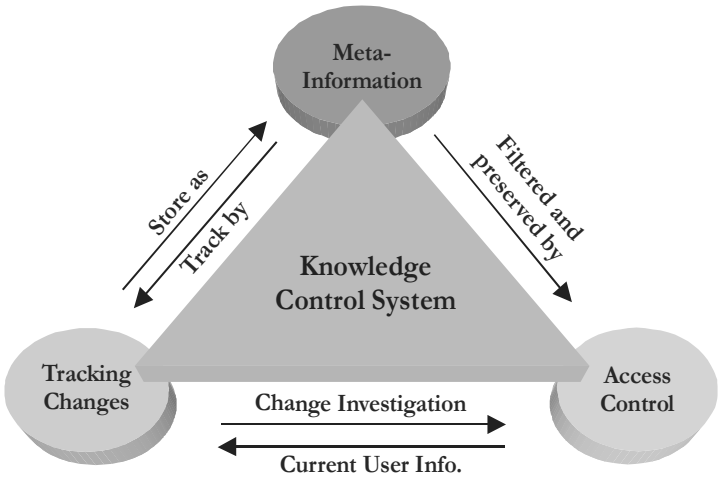
Abstract. The real-world knowledge management applications require features such as versioning and fine-grained access control. Each of them raises the issue of tracking the changes in a knowledge base. Important part of the research presented is the definition of a formal model for tracking changes in graph-based data models. It was used in the ontology middleware module developed under the On-To-Knowledge project as an extension of the Sesame RDF(S) repository. This paper is further development of the results reported in [5].

1. Introduction

The following features are considered critical for development, management, maintenance, and use of middle-size and big knowledge bases:

- Versioning (tracking changes) of knowledge bases;
- Access control (security) system;
- Meta-information for knowledge bases.

These three aspects are tightly interrelated among each other as depicted on the following scheme. The dependencies are explained in the corresponding sections.



The composition of the three functions above represents a Knowledge Control System (KCS) that provides the knowledge engineers with the same level of control and manageability of the knowledge in the process of its development and maintenance as the source control systems (such as CVS) provide for the software. From the perspective of the end-user applications, KCS can be seen as equivalent to the database security, change tracking and auditing systems. Our KCS is carefully designed to support these two distinct use cases.

The work presented here was carried as part of the On-To-Knowledge project. The design and implementation of the change tracking within the ontology middleware module presented here is an extension of the Sesame architecture (see [1]). Earlier stage of this research is presented in bigger details in [5] where the reader can find more about the Access control system (security), which is out of the scope of this paper.

In the rest of this introductory section we define better the scope of our research and the terminology used. Section 2 is dedicated to the model and principles for tracking changes in RDF(S) repositories. The implementation approach of is presented in Section 3. A short conclusion follows in the last section.

1.1. Ontologies vs. Knowledge Bases

A number of justifications in the terminology are necessary. An almost trivial but very important question is ”What the KM tools support: ontologies, data, knowledge, or knowledge bases?” Due to the lack of space we are no going in to comment this basic notions here. A simple and correct answer is ”All of this”. The ontology middleware module extends the Sesame RDF(S) repository that affects the management of both ontologies and instance data in a pretty much unified fashion.

For the purpose of compliance with Sesame, here the term repository is used to denote a compact body of knowledge that could be used, manipulated, and referred as a whole. Such may contain (or host) both ontological assertions and instance data.

1.2. Versioning vs. Tracking Changes

The problem for tracking changes within a knowledge base is addressed in this section. It is important to clarify that higher-level evaluation or classification of the updates (considering, for instance, different sorts of compatibility between two states or between a new ontology and old instance data) is beyond the scope of this work. Those are studied and discussed in depth in [2], sub-section 2.2. The tracking of the changes in the knowledge (as discussed here) provides the necessary basis for further analysis. In summary, the approach taken can be shortly characterized as ”versioning of RDF on a structural level in the spirit of the software source control systems”.

1.3. Related Work

Here we will shortly comment several studies related to versioning of a complex data objects. Although some of the sources discuss similar problems there is not one

addressing ontology evolution and version management in a fashion allowing granularity down to the level of statements (or similar constructs) and capturing of the interactive changes in knowledge repositories such as assertions and retractions.

Database schema evolution and the tasks related to keeping schema and data consistent to each other can be recognized as a very similar problem. A detailed and pretty formal study on this problem can be found in [3, 4] – it presents an approach allowing the different sorts of modifications of the schema to be expressed within suitable description logic.

2. Versioning Model for RDF(S) Repositories

A model for tracking of changes, versioning, and meta-information for RDF(S) repositories is proposed, i.e. (i) the knowledge representation paradigm supported is RDF(S) and (ii) what is being tracked are repositories – independently from the fact if they contain ontologies, instance data, or both. The decision to support tracking of changes, versioning, and meta-information for RDF(S) repositories has a number of consequences and requires more decisions to be taken. The most important principles are presented in the next paragraphs.

VPR1: *The RDF statement is the smallest directly manageable piece of knowledge.*

Each repository, formally speaking, is a set of RDF statements (i.e. triples) – these are the smallest separately manageable pieces of knowledge. There exist arguments that the resources and the literals are the smallest entities – it is true, however they cannot be manipulated independently – they always appear as a part of a triple. To summarize, there is no way to add, remove, or update (the description of) a resource without also changing some statements, while the opposite does not hold.

VPR2: *An RDF statement cannot be changed – it can only be added and removed.*

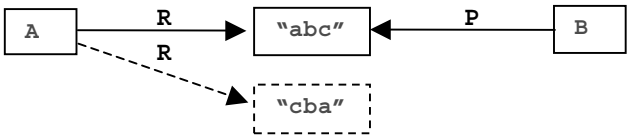
As far as the statements are nothing more than triples, changing one of the constituents, just converts it into another triple. It is because there is nothing else but the constituents to determine the identity of the triple, which is an abstract entity being fully defined by them. Let us take for instance the statement **`ST1=<A, PR1, B>`** and suppose B is a resource, i.e. an URI of resource. Then ST1 is nothing more but a triple of the URIs of A, PR1, and B – if one of those get changed it will be already pointing to a different resource that may or may not have something in common with the first one. For example, if the URI of A was **`http://x.y.z/o1#A`** and it get changed to **`http://x.y.z/o1#C`** then the statement **`ST2=<C,PR1,B>`** will be a completely different statement.

Further, if the resource pointed by an URI gets changed two cases could be distinguished:

- The resource is changed but its meta-description in RDF is not. Such changes are outside the scope of the problem for tracking changes in formally represented knowledge, and particularly in RDF(S) repositories.
- The description of the resource is changed – it can happen iff a statement including this resource get changed, i.e. added or removed. In such case, there is another

statement affected, but the one that just bears the URI of the same resource does not.

There could be an argument, that when the object of a triple is a literal and it gets changed, this is still the same triple. However, if there is for instance statement **<A,R,"abc">** and it changes to **<A,R,"cba">**, the graph representation shows that it is just a different arc because the new literal is a new node and there could be other statements (say, **<B,P,"abc">**) still connected to the old one.
As a consequence here comes the next princple:



VPR3: *The two basic types of updates in a repository are addition and removal of a statement*

In other words, those are the events that necessarily have to be tracked by a tracking system. It is obvious that more event types such as replacement or simultaneous addition of a number of statements may also be considered as relevant for an RDF(S) repository change tracking system. However, those can all be seen as composite events that can be modeled via sequences of additions and removals. As far as there is no doubt that the solution proposed should allow for tracking of composite events (say, via post-processing of the sequence of the simple ones), we are not going to enumerate or specify them here.

VPR4: *Each update turns the repository into a new state*

Formally, a state of the repository is determined by the set of statements that are explicitly asserted. As far as each update is changing the set of statements, it is also turning the repository into another state. A tracking system should be able to address and manage all the states of a repository.

2.1. History, Passing through Equivalent States

The history of changes in the repository could be defined as sequence of states, as well, as a sequence of updates. It has to be mentioned that in the history, there could be a number of equivalent states. It is just a question of perspective do we consider those as one and the same state or as equivalent ones. Both perspectives bear some advantages for some applications. We accepted that there could be equivalent states in the history of a repository, but they are still managed as distinct entities.

2.2. Versions are labeled states of the repository

Some of the states of the repository could be pointed out as versions. Such could be any state, without any formal criteria and requirements – it completely depends on the user's or application's needs and desires. Once defined to be a version, the state becomes a first class entity for which additional knowledge could be supported.

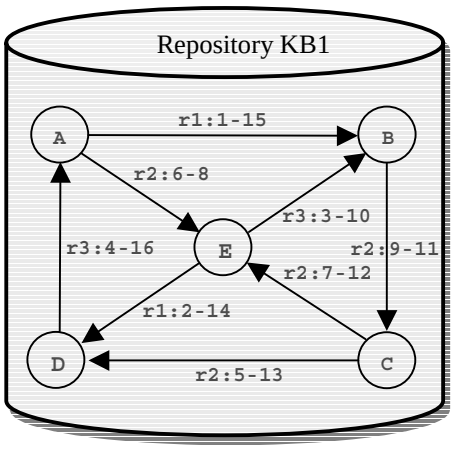
3. Implementation Approach

For each repository, there is an *update counter* (UC) – an integer variable that increases its value each time when the repository is updated. Let us call each separate value of the UC *update identifier, UID*. Then for each statement in the repository the UIDs when it was added and removed are known – these values determine the "lifetime" of the statement. It is also the case that each state of the repository is identified by the corresponding UID. For each state it is straightforward to find the set of statements that determine it – those that were "alive" at the UID of the state being examined.

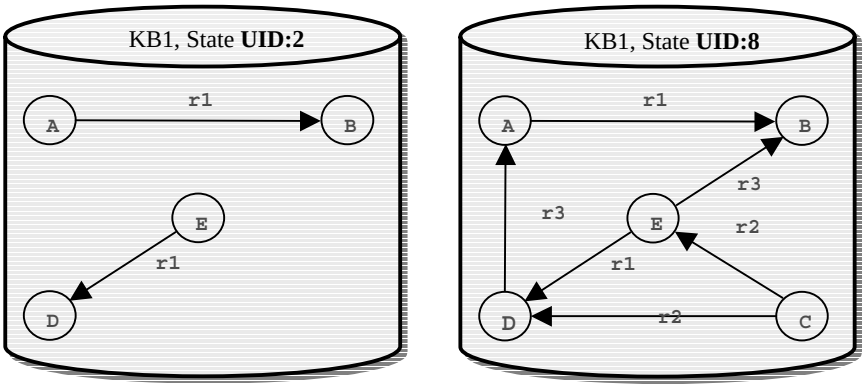
The approach could be demonstrated with the sample repository KB1 and its "history". The repository is represented as a graph where the lifetime of the statements is given separated with semicolons after the property names. The history is presented via events in format: **`UID:nn {add|remove} <subj, pred, obj>`**

History:

```
UID:1 add <A, r1, B>
UID:2 add <E, r1, D>
UID:3 add <E, r3, B>
UID:4 add <D, r3, A>
UID:5 add <C, r2, D>
UID:6 add <A, r2, E>
UID:7 add <C, r2, E>
UID:8 remove <A, r2, E>
UID:9 add <B, r2, C>
UID:10 remove <E, r3, B>
UID:11 remove <B, r2, C>
UID:12 remove <C, r2, E>
UID:13 remove <C, r2, D>
UID:14 remove <E, r1, D>
UID:15 remove <A, r1, B>
UID:16 remove <D, r3, A>
```



Here follow two "snapshots" of states of the repository respectively for UIDs 2 and 8



It is an interesting question how we handle in the above model, multiple additions and removals of one and the same statement, which in a sense periodically appears and disappears form the repository. We undertake the approach to consider them as separate statements, because of reasons similar to those presented for the support of distinguishable equivalent statements.

4. Conclusion and future work

The ontology middleware, part of which is the tracking changes module presented still have to prove itself in real-world applications. At this stage it is work in progress inspired by the methodology, tools, and case studies of the On-To-Knowledge project.

References

1. Jeen Broekstra, Arjohn Kampman. *Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema.*Deliverable 9, On-To-Knowledge project, October 2001. http://www.ontoknowledge.org/downl/del10.pdf
2. Ying Ding, Dieter Fensel, Michel Klein, Borys Omelayenko. *Ontology management: survey, requirements and directions*. Deliverable 4, On-To-Knowledge project, June 2001. http://www.ontoknowledge.org/downl/del4.pdf
3. Enrico Franconi, Fabio Grandi, Federica Mandreoli. *Schema Evolution and Versioning: a Logical and Computational Characterization.* In "Database schema evolution and meta-modeling" - Ninth International Workshop on Foundations of Models and Languages for Data and Objects, Schloss Dagstuhl, Germany, Sept.18-21, 2000. LNCS 2065, pp 85-99
4. Enrico Franconi, Fabio Grandi, Federica Mandreoli. *A Semantic Approach for Schema Evolution and Versioning of OODB.* Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, August 17-19, 2000. pp 99-112
5. Atanas Kiryakov, Kiril Iv. Simov, Damyan Ognyanov. *Ontology Middleware: Analysis and Design.* Deliverable 38, On-To-Knowledge project, March 2002.
6. W3C; Ora Lassila, Ralph R. Swick, eds. *Resource Description Framework (RDF) Model and Syntax Specification.* http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/