Supporting Complex Changes in RDF(S) Knowledge Bases

Theodora Galani[1+], Yannis Stavrakas[1*], George Papastefanatos[1+], Giorgos Flouris[2*]

[1]Institute for the Management of Information Systems, RC ATHENA, GREECE
[2]Institute of Computer Science, FORTH, GREECE
{theodora, yannis, gpapas}@imis.athena-innovation.gr,
fgeo@ics.forth.gr

Abstract. The dynamic nature of web data brings forward the need for maintaining data versions as well as identifying semantically rich changes between them. In this paper, we advocate the need for supporting complex changes in evolving RDF(S) knowledge bases. We outline the basic challenges and provide solution insights through a real-world example from the field of biology.

Keywords: change management, data evolution, rdf(s)

1 Introduction

The increasing amount of information published on the web poses new challenges for data management. A central issue concerns evolution management, as the dynamic nature of data brings forward the need for maintaining data versions as well as identifying changes between them. For example, biologists often use ontologies in order to curate their data from multiple domains of interest like anatomy, diseases, biomedical investigations, etc. These ontologies are frequently updated as errors may need to be fixed or new knowledge about the state of the art may need to be incorporated. As a result, curators of depending ontologies are interested in understanding the evolution history in order to learn more about the changes that have taken place on the respective domain of interest.

In this paper, we argue that understanding data evolution should involve high-level, semantically rich, user-defined changes that we call *complex changes*. Formalizing complex changes involves facing the challenges of modeling, defining, detecting and querying changes. Although the concept of complex changes is not bound to any specific data model, in this paper we focus on RDF(S) knowledge bases, as RDF is a de-facto standard for representing data on the web. The goal of this paper is to highlight the main challenges as well as possible solution insights towards a framework that makes changes first class citizens.

* Supported by the EU-funded ICT project "DIACHRON" (agreement no 601043).
+ Supported by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

The paper outline is as follows. In section 2 we discuss in detail the challenges for supporting complex changes. In section 3 we provide an end-to-end real world example that demonstrates important aspects of our approach to the aforementioned problems. Finally, in section 4 we conclude the paper.

2 Challenges and Roadmap

Modeling changes. An approach for modeling changes in RDF(S) knowledge bases would be determining the added and deleted triples between versions. However, this is not sufficient for understanding data evolution. Human-readable, high-level changes should be employed. In this case, two basic issues must be taken into consideration.

- Granularity of changes. Fine-grained or coarse-grained changes? Fine-grained changes have the advantage of describing primitive changes, while coarse-grained changes provide more semantics and conciseness by grouping primitive changes in logical units.
- Semantics of changes. Model-specific or data- and application- specific changes? Model-specific changes describe modifications that may appear in a specific representation model. They constitute a fixed set of generic changes. On the other hand, data- or application- specific changes represent user-defined changes that suit on specific use-case scenarios. Supporting user-defined changes has the advantage of allowing different interpretations of evolution.

In order to tackle the above issues, we distinguish between *simple* and *complex* changes. Simple changes constitute a fixed set of fine-grained, model-specific changes. Complex changes are coarse-grained, user-defined, application-specific changes.

In previous works [2, 4, 6, 10], various lists of predefined changes have been proposed, usually distinguished into fine-grained and coarse-grained changes. In [6] formal semantics are defined guaranteeing useful properties. In [1] an approach for modeling changes as sequences of triples is proposed.

Defining changes. A declarative language for defining changes is needed for supporting user-defined complex changes. The language expressiveness should be investigated. A complex change definition should consist of a finite, non-empty list of simple or (already defined) complex changes, and a set of constraints over these changes. The supported constraints may filter parameter values, express pre- or post- conditions, relate change parameters, pose cardinality constraints (e.g. there must be at least one change of a specific type) and allow or not overlaps among changes.

In [8] a language for defining high-level changes, called Change Definition Language, has been proposed. Defined changes are detected over a version log [7] using temporal queries, assuming that the version log is populated as modifications apply. In [9] a framework for defining changes using SPARQL query features is presented as an extension of [6].

Detecting changes. As new dataset versions are periodically released, simple and complex changes can be detected among versions. In [4] a fixed-point algorithm for comparing ontology versions has been proposed. The algorithm is based on heuristic-

based matchers, introducing uncertainty to the results. On the other hand, in [6] the detection process does not introduce any uncertainty to the results. In our approach, we need to identify the rules for mapping complex change definitions into processes that return instances of the respective change patterns. The performance of the detection process has to be investigated with respect to the number of changes and the type of constraints in complex change definitions, as well as the dataset versions' size and the number of changes performed between them.

Querying changes. In our view, querying data evolution should be based on data as much as on changes. Changes, like data, can appear in the query body to express complex conditions, like the fact that an entity has been modified in a specific manner, or can be returned by the query in order to retrieve explicit change instances. Some interesting query types that should be supported are the following:

- Retrieve changes among versions, or restrict selected changes by the type of change or the elements that they have affected or the versions between which they are detected.
- Retrieve elements, given that changes of specific type have affected them at specific versions.

3 An End-to-End Example

The Experimental Factor Ontology (EFO) [3] provides a systematic description of many data elements available in EBI[1] databases, and for external projects. It combines parts of several biological ontologies regarding anatomy, disease and chemical compounds in order to support data annotation, analysis and visualization. EFO is frequently updated as new classes are added, while others are changed or made obsolete. Classes in EFO are described by metadata like class label, definition, synonyms, etc.

Consider that a new class is added into the ontology. This class is also assigned with a class label, a textual definition and synonyms of the class label. The class label corresponds to rdfs:label annotation property, the textual definition corresponds to the efo:definition property and the synonym to the efo:alternative-term property. Note that for simplicity and space limitations we consider only these operations.

Modeling changes. These changes are fine-grained and can be described by model-specific operations. The addition of a new class can be modeled as *Add_Type_Class(c)*, where *c* is the new class. The addition of a new label can be modeled as *Add_Label(c, l)*, where *c* is the respective class holding the new label *l*. The addition of a new definition or synonym corresponds to an addition of a new property and can be modeled as *Add_Property_Instance(s, p, o)*, where *p* is the new property which is assigned to class *s* with value *o*. In our approach, these are simple changes. We can rely on [6] for defining simple changes by selecting a minimal set of primitive changes on RDF(S) having the properties of completeness and unambiguity.

Notice that *Add_Property_Instance* suits all possible properties, while in this scenario the assigned properties are of two specific types: efo:definition and

[1] http://www.ebi.ac.uk/

efo:synonym. It is more suitable to have intuitive changes regarding the specific properties involved, like *Add_Definition* and *Add_Synonym*. Also, the discussed modifications are likely to appear jointly. As a result, it may be useful to demonstrate these changes as a unit. Therefore, they can be grouped into one change named *Add_Annotated_Class*. In our approach, these are examples of complex changes.

Defining changes. The complex changes *Add_Definition*, *Add_Synonym* and *Add_Annotated_Class* can be defined as follows:

```
CREATE COMPLEX CHANGE Add_Definition(class, definition) {
   CHANGE LIST Add_Property_Instance(class, prop, definition);
   SELECTION FILTER prop='efo:definition';   };
CREATE COMPLEX CHANGE Add_Synonym(class, synonym) {
   CHANGE LIST Add_Property_Instance(class, prop, synonym);
   SELECTION FILTER prop='efo:alternative_term';   };
CREATE COMPLEX CHANGE Add_Annotated_Class(class, label, defini-
tion, synonym) {
   CHANGE LIST Add_Type_Class(class), Add_Label(class, label),
Add_Definition(class, definition), Add_Synonym(class, synonym)
*;   };
```

The name and parameters of each defined complex change are declared right after the *CREATE COMPLEX CHANGE* clause. In the *CHANGE LIST* clause the contained simple or complex changes are declared. Note that the asterisk (*) beside *Add_Synonym* in *Add_Annotated_Class* definition indicates that there might be zero, one, or more such changes, one for each added synonym, posing a cardinality constraint. Defining *Add_Definition* and *Add_Synonym* includes a constraint, declared in the *SELECTION FILTER* clause, filtering the property type. In *Add_Annotated_Term*, the parameter name *class* is used among the contained changes, indicating that they refer to the same actual class.

Detecting changes. As ontology versions are periodically released, we can identify the changes that have occurred among versions. Simple changes have to be detected first. Notice that *Add_Definition* and *Add_Synonym* are defined in terms of simple changes, while *Add_Annotated_Class* includes complex changes too. Therefore, *Add_Definition* and *Add_Synonym* should be detected first by evaluating their definitions over the detected simple change instances, while *Add_Annotated_Class* next as it depends on complex change instances too. Alternatively, *Add_Annotated_Class* can be expressed in terms of simple changes, by substituting *Add_Definition* and *Add_Synonym* changes with their definitions. In this way, all complex change definitions can be evaluated over the detected simple change instances. The detected simple and complex change instances constitute a hierarchy of changes, where the user can see the changes themselves as well as how they are interconnected.

Querying changes. For querying changes, SPARQL can be extended with suitable keywords. The following query gives an example of querying changes. It returns all classes that have been added and annotated between versions 2.45 and 2.46. For this example, we assume that defined changes and detected instances are represented in an ontology of changes as in [9]. Notice that the requested classes are the value of

co:aac_p1 parameter of *Add_Annotated_Class*. Also, *change_span* is a function that verifies whether the complex change instance (*?c*) is detected between the requested versions. Finally, the *FROM CHANGES ON DATASET* clause declares that the triples pattern concerns changes regarding a specific dataset <*D*>.

```
SELECT ?class
FROM CHANGES ON DATASET <D>
WHERE {
  ?c rdf:type co:Add_Annotated_Class; co:aac_p1 ?class.
  FILTER change_span(?c BETWEEN VERSION 2.45 AND 2.46). }
```

4 Conclusions

In this paper we advocated the need for formalizing complex changes over RDF(S) knowledge bases and outlined the basic challenges that have to be faced to realize our vision. An example inspired from the biological domain is used to motivate the need for complex changes and present the basic concepts of a possible solution. Nevertheless, supporting complex changes may be useful in any evolving domain.

5 References

1. S. Auer and H. Herre. A versioning and evolution framework for RDF knowledge bases. In Perspectives of Systems Informatics, 6th International Andrei Ershov Memorial Conference on, 2007.
2. M. Klein. Change management for distributed ontologies. Ph.D. thesis, Vrije University, 2004.
3. J. Malone, E. Holloway, T. Adamusiak, M. Kapushesky, J. Zheng, N. Kolesnikov, A. Zhukova, A. Brazma, H. Parkinson. Modeling Sample Variables with an Experimental Factor Ontology. Bioinformatics 26(8):1112-1118, 2010.
4. N. F. Noy, and M. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In Proceedings of the 18th National Conference on Artificial Intelligence, 2002.
5. G. Papastefanatos, Y. Stavrakas, and T. Galani. Capturing the history and change structure of evolving data. 7th International Conference on Advances in Databases, Knowledge, and Data Applications, 2013.
6. V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. High-level change detection in RDF(S) KBs. ACM Trans. Database Syst., 38(1), 2013.
7. P. Plessers and O. De Troyer. Ontology change detection using a version log. In Proceedings of the 4th International Semantic Web Conference, 2005.
8. P. Plessers, O. De Troyer, and S. Casteleyn. Understanding ontology evolution: A change detection approach. J. Web Sem. 5(1): 39-49, 2007.
9. Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, and Y. Stavrakas. A flexible framework for defining, representing and detecting changes on the data web. CoRR abs/1501.02652, 2015.
10. L. Stojanovic. Methods and tools for ontology evolution. Ph.D. thesis, University of Karlsruhe, 2004.