SEMVERSION:
AN RDF-BASED ONTOLOGY VERSIONING SYSTEM

Max Völkel
FZI / Universität Karlsruhe
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
voelkel@fzi.de

Tudor Groza
DERI, National University of Ireland,
Galway, Ireland
tudor.groza@deri.org

ABSTRACT

Knowledge domains and their formal representations via ontologies are typically subject to change in practical applications. Additionally, engineering of ontologies often takes place in distributed settings where multiple independent users interact. Therefore, change management for ontologies becomes a crucial aspect for any kind of ontology management environment. This paper introduces a new RDF-centric versioning approach and an implementation called SemVersion. SemVersion provides structural and semantic versioning for RDF models and RDF-based ontology languages like RDFS.

KEYWORDS

Knowledge management, Ontology versioning, Triple store, RDF.

1. INTRODUCTION

For many practical applications, ontologies (cf. Staab and Studer 2004) can not be seen as static entities, they rather change over time. Support for change management is crucial to support uncontrolled, decentralized and distributed engineering of ontologies. In order to handle ontology modifications in time, a versioning system is needed, to keep track of changes and versions. First approaches have been described in (Klein 2004 and Stojanovic 2004). But, there is no tool that functions as a standard versioning system for ontologies like CVS does in the field of software development.

This paper introduces an RDF-based approach that provides versioning for RDF models and RDF-based ontology languages like RDFS, OWL flavors or TRIPLE (Sintek and Decker 2002). We present a working methodology accompanied by its implementation in the system SemVersion. The methodology and the system provide a well-defined core functionality for ontology versioning.

Our approach is inspired by the classical CVS system for version management of textual documents (e.g. Java code). The core element of our approach is the separation of language-specific features (i.e. the diff) from general features (such as structural diff, branch and merge, management of projects and metadata). A specialty of RDF is the usage of so-called blank nodes. As part of our approach we present a method for *blank node enrichment* which is required for the versioning of such blank nodes.

The paper is structured as follows. Section 2 provides an overview of the related work in the domain. We continue then by describing the general idea behind our system in Section 3 and going into implementation details in Section 4. Section 5 shows how SemVersion can be used and in the end the paper presents our conclusion (Section 6).

2. RELATED WORK

The very popular concurrent version system (CVS) initially was a collection of scripts to simplify the handling of the revision control system (RCS). RCS operates in a file-centric way by using a "lock-modify-unlock"-style. However, CVS works on the syntactical level, not on the conceptual level. I.e., it is not capable of versioning objects and in particular not capable of versioning ontological entities and their complex structure. The underlying diff operation is only capable of showing the syntactical differences between two files (based on the differences of text lines).

Following terminology from the database community (cf. Roddick 1995) we mainly distinguish between *ontology versioning* and *ontology evolution*. The difference between schema evolution and ontology evolution is shown in (Noy and Klein 2003). Ontology versioning is accommodated when an ontology management system allows for handling of ontology changes by creating and managing different versions of it. Ontology evolution is accommodated when an ontology management system facilitates the modification of an ontology by preserving its consistency (e.g. Stojanovic 2004).

A first survey on causes and consequences of changes in an ontology is presented in (Klein and Fensel 2001), followed by an implementation for ontology versioning (Klein and Fensel 2002) that is based on the comparison of two ontology versions in order to detect changes. Basically, the system compares ontological classes, displays them side-by-side in RDF/XML and leaves it to the user to state "identical" or "conceptual change.

Another example of developed, or under development, ontology versioning tool is (KPOntology). It is similar with our system, being a library for versioning ontologies and allowing the use of different triple stores. The big difference is given by the fact that KPOntology provides a separate library for each triple store, while our approach is unified. Only the programmer using SemVersion will be aware of the triple store underneath (by changing one line of code), but to the end user, this will be transparent.

Among the related fields we can include schema evolution in databases (cf. e.g. Huersch 1997 and Barnejee el al. 1987) and evolution of XML documents (cf. e.g. Su et al. 2002 and Cobena 2003). Along with a critical discussion of the relationships between ontology evolution and these approaches one can also find a quite extensive amount of additional related work in (Stojanovic 2004).

The aim of this paper is a cost-efficient ontology versioning. Therefore we identify reusable parts of a versioning system and explain where the system needs to be tailored to a particular ontology language. We emphasize re-use of existing components as much as possible, such as the RDFS reasoning offered by libraries like Jena. This allowed us to keep the code in SemVersion small and manageable.

3. RDF-LAYERED ONTOLOGY VERSIONING METHODOLOGY

The most elementary modeling primitive that is needed to model a shared conceptualization of some domain is a way to denote entities and to unambiguously reference them. For this purpose RDF uses URIs, identifiers for resources, which are supposed to be globally unique. Every ontology language needs to provide means to denote entities. For global systems, the identifier should be globally unique. Having entities that can be referenced, the next step is to describe relations between them. As relations are semantic core elements, they should also be unambiguously addressable. Properties in RDF can be seen as binary relations. This is the very basic type of relations between two entities. More complex types of relations (e.g. OWL class restrictions) can be modeled by defining a special vocabulary (possibly using auxiliary nodes) for this purpose on top of RDF, like it has been done in OWL.

The various ontology languages differ in their vocabulary, their logical foundations, and epistemological elements, but they have in common that they describe structures of entities and their relations. Therefore RDF is the largest common denominator of all ontology languages. RDF is not only a way to encode the ontology languages or just an arbitrary data model, but it is a structured data model that matches exactly the structure of ontology languages. The serialization of RDF as RDF/XML is completely irrelevant for our approach.

3.1 Management aspects of versioning

Our general idea is the re-use of data management functionality across ontology languages. The relations between different versions of an RDF model or ontology are the same, regardless of the semantics used. Data management deals with storage and retrieval of chunks of data. In our case, the smallest unit of data we store and retrieve is a model (also called 'triple set'). A model is a set of RDF triples. A versioned model consists of a triple set for the content plus an arbitrary number of statements about this model. We thus call this *model based versioning* in contrast to *statement based versioning*.

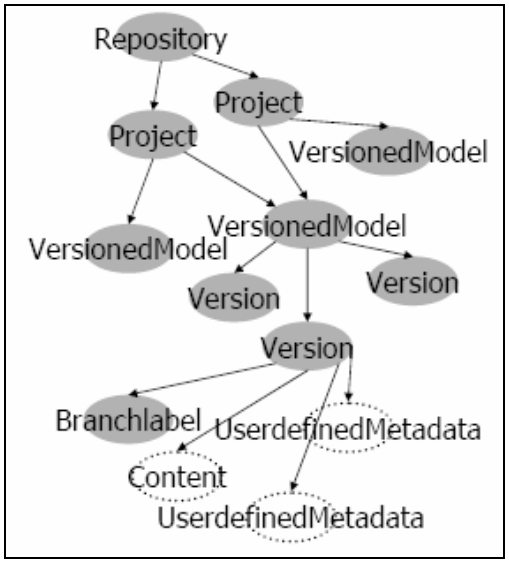

Fig. 1 Data Model for RDF Versioning

SemVersion's data model is depicted in **Error! Reference source not found.**. It has a **repository of projects**. They can be created, listed and removed from that repository. A project can hold a number of **versioned models**. A versioned model is the container for a single RDF model or ontology under version control. It has a **root version** and also knows all other **versions** that are direct or indirect descendants of the root version. Versioned models are quite an important concept and give the user the ability to retrieve the right version by e. g. listing all branches or simple getting the most current "main" branch version.

A **version** is the most central concept. It is a model decorated with metadata. A version knows where it comes from (it parents), has a branch, a label and optionally even a comment and a provenance URI. This meta-information about versions can be managed independent of the versioned artifacts themselves. Thus this management layer can be designed very flexible and reusable. The user can commit a **model** as the successor of a version; create a new version by merging two existing models or commit a diff. Committing diffs is useful, if the models become really large and change only little.

Users can store arbitrary RDF encoded metadata objects for each project, versioned model and most important for each version. This data is stored in the RDF storage layer and linked by RDF statements to the versioning artifact it belongs to. Metadata models are also URI-addressable. This metadata strategy enables a good re-use of the SemVersion system, as e. g. the evolution log of an ontology engineering tool could be assigned to a version with this mechanism.

3.2 Functional aspects of versioning

Versioning functionality deals with ontology language specific functionality such as the structural diff (ignoring semantics) and the semantic diff (depends on ontology language; uses structural diff). Other functional aspects discussed in this section are blank nodes, which make the issue of comparing versions harder and the branch and merge operations.

3.2.1 Structural vs. semantic diff

Although the structural diff is the same for all ontology languages, we describe it here for sake of consistency. The structural diff is simply the set-theoretic difference of two RDF triple sets. Libraries such as (JENA) have built-in functions to compute this set-difference. RDF2GO (Voelkel 2005) also offers a native implementation.

The diff function $d(A,B) \rightarrow \langle a(A,B), r(A,B) \rangle$ is a non commutative function from two triple sets (A, B) to two triple sets of added ($a(A,B)$) and removed ($r(A,B)$) statements, with $a(A,B) = B - A = B \setminus (A \cap B)$ and $r(A,B) = A - B = A \setminus (A \cap B)$. Such diffs can be computed by simple set arithmetic for triple sets that contain only URIs and literals, as shown in (Kiryakov 2002). Blank nodes cause some problems here, as discussed in the next section.

version A:	version B:
a rdfs:type c b rdfs:type c c rdfs:subClassOf d	a rdfs:type d b rdfs:type d c rdfs:subClassOf d e rdfs:type d
added:	**removed:**
a rdfs:type d b rdfs:type d e rdfs:type d	a rdfs:type c b rdfs:type c

Fig. 2. Example of a structural diff

The semantic difference has to take the semantics of the used ontology language into account. It is therefore not possible to write a generic algorithm for this. An intuitive way to understand the concept of a semantic diff goes like this: Let's assume we have RDF Schema as our ontology language. Further we have two models A and B, which express two versions of an RDF Schema. In order to compute the semantic diff, we use RDF Schema entailment on model A and infer all triples we can (*Inf*(A)). Then we do the same for model B (*Inf*(B)). In the end we calculate a structural diff on *Inf*(A) and *Inf*(B). This is not the same as the structural diff between model A and B. Note also that this diff operator is ontology language specific and the proposed algorithm does not work for OWL.

version A:	version B:
a rdfs:type c b rdfs:type c c rdfs:subClassOf d	a rdfs:type d b rdfs:type d c rdfs:subClassOf d e rdfs:type d
added:	**removed:**
e rdfs:type d	

Fig. 3. Example of a semantic diff under RDFS semantics

A possible way to compute a semantic diff in RDFS is thus to materialize the complete entailment (transitive closure) and then perform a structural diff, like it is done in SemVersion. For RDF Schema the calculation of the transitive closure can be re-used from the Jena framework (JENA). However, in certain cases this might not be realizable, especially when the models have an exponential growth. The calculation of a semantic diff can be accomplished by a language specific reasoner or by a language specific set of rules. These rules can be formulated in a language like TRIPLE as demonstrated in (Sintek and Decker 2002). Initially we provide support for RDF Schema. If the structural diff of two models is empty, then the semantic diff must also be empty. The inverse is not necessarily true: There might be two different RDF models which encode the same semantic model. RDFS reasoning is well explored, so SemVersion reuses existing software components for this task.

3.2.2 Blank nodes

Blank nodes cause problems in computing the structural diff, as we have no knowledge about the relation (equal or not) between two blank nodes from different models. The RDF semantics dictates to treat them as different. In a versioning context, this leads to the unwanted fact that the diff between a model and itself is not empty, if it contains blank nodes.

version A:	version B:
_:1 :hasName “Max”	_:3 :hasName “Max” _:3 :hasPhone “123” _:5 :hasName “Max” _:5 :hasPhone “456”
added:	**removed:**
_:3 :hasName “Max” _:3 :hasPhone “123” _:5 :hasName “Max” _:5 :hasPhone “456”	_:1 :hasName “Max”

Fig. 4. Example of too large diffs when comparing models with blank nodes

As a work-around we invented the concept of blank node enrichment, which attaches artificial inverse functional properties to every blank node. This brings no changes to the RDF semantics and helps to identify equal blank nodes across models. Most RDF processing tools will leave this information intact. In SemVersion, the content of every version is blank node enriched before it is stored in the RDF storage layer.

As blank nodes are existential variables, one can only assume blank node equality between two RDF graphs, if exactly the same statements are made about them. For versioning we thus suggest to either use blank node enrichment or avoid blank nodes at all.

Blank nodes are used in practice for e.g. in OWL for property restrictions or in FOAF to denote persons.

3.2.3 Branch and Merge

Branch and merge operations allow ontology engineers to follow multiple development paths in parallel. Users can create branches from each version in the version tree. A new branch is create like a new version, but with an additional branch label for the new branch. It is possible to merge arbitrary versions, not only those at the end of a branch. A merge of two versions is simply the set union of the triple sets.

Merging two branches is different. First we look at the most recent common version of the two branches (let's name it *c*). Such a version always exists, as branches can only be created by committing a version to an existing version. We also take two versions from the different branches, in most cases the most recent ones (*a* and *b*). Consider the sample version tree provided in Fig. 5. Here c = A, a = A”, b = B”. In order to merge *b* back into *a*, we compute the **diff**(*c*, *b*) and apply it to *a*.

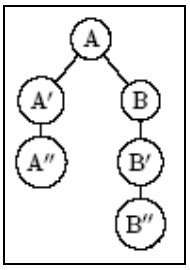

Fig. 5. A sample version tree

3.2.4 Conflict detection

RDF models themselves are never in a conflict state. But a diff between two models can indicate a conflict on the ontology layer. SemVersion uses a simple conflict detection heuristic, that detects if a diff adds statements about a resource that was present in *c*, but has been removed on its way to *a*. This means, the URI of a resource was used in triples from *c*, but no triple in *a* contains this URI.

4. IMPLEMENTATION

The aim of our implementation was to create a pure Java library. Additional HTTP-based web services will be added as required.

The high-level architecture of the implementation consists of several layers (Fig.). Each layer depends on the layer below. To users only the SemVersion API and the RDF2GO API are exposed. The layers are:

- **SemVersion API** – the versioning API to be used by users;
- RDFREACTOR – a framework for domain-specific object-oriented RDF access in Java;
- RDF2GO – an abstraction over triple (and quad) stores;
- YARS – a scalable quad store.

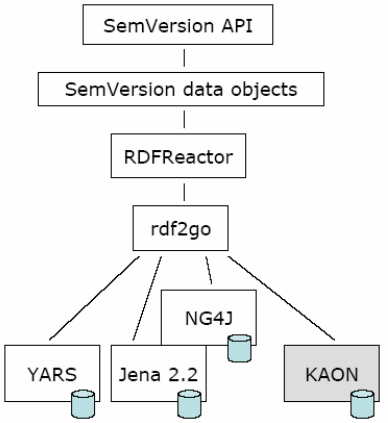

Fig. 6. The layered architecture of SemVersion

Following, we will explain the methods used by us to deal with some of the ontology versioning issues:

- **Storage layer access** – an ontology versioning system should scale in many dimensions. There already exist scalable RDF stores including also remote query and update functionality. SemVersion uses RDF2GO which abstracts away the triple store implementation and gives the user a simple Java-centric API for model changes.
- **Handling commits** – the new version will simply be sent to the RDF store having a globally unique generated URI. This guarantees that the retrieval will give the user back what she checked in.
- **Generating globally unique URIs** – The strategy for generating globally unique URIs is as follows: (i) SemVersion server's URL is added as first part of the URI; (ii) we then append the current system time, measured in milliseconds; (iii) the last part of the URI will be represented by an internal counter, in order to be able to issue different URIs in one particular moment of time. The URI generator cannot guarantee uniqueness, but the likelihood for the same URI being generated twice is really low.

5. USING SEMVERSION FOR ONTOLOGY VERSIONING

In this section we explain how SemVersion can be used to build an ontology versioning system for a particular RDF-based ontology language. SemVersion is currently used in the MarcOnt portal[1]. For *any* RDF-based ontology language, we can reuse the complete version data management infrastructure of SemVersion, which includes managing projects, versioned models, versions and metadata for each of these concepts. Some basic versioning functions can also be used out-of-the box such as retrieve, commit and branch.

The following language specific versioning functions need to be implemented additionally:

- **Semantic diffs** – the basic structural diff provided by SemVersion (for RDF/S) is not identical with the semantic diff. To calculate the semantic diff d_l a system has to know the semantics of the specific language *l*. The semantic closure $s_l(M)$ of a model *M* is the set of all statements that can be concluded from the statements in *M* under the semantics of the RDF-based ontology language *l*.

[1] http://www.marcont.org

The semantic diff of two models A and B is $d_l(A,B) \rightarrow \langle a_l(A,B), r_l(A,B) \rangle$ with $a_l(A,B) \rightarrow s_l(B) \setminus (s_l(A) \cap s_l(B))$ and $r_l(A,B) \rightarrow s_l(A) \setminus (s_l(A) \cap s_l(B))$. The calculation of a semantic diff can be accomplished by a language specific reasoner or by a language specific set of rules. These rules can be formulated in a language like TRIPLE as demonstrated in (Sintek and Decker 2002).

- **Semantic conflict detection** – the ontology language semantics determine what should be considered as a conflict e. g. if the result of a merge is an inconsistent ontology.

Additionally, the specific ontology versioning system might want to have a special diff encoding. Our system can be adopted by providing mapping rules between the RDF diff and the specific language encoded diff. Further a specific versioning system can use the 'user defined metadata' functionality of SemVersion for storing specific metadata like access rights, degree of agreement, mappings between versions etc.

5.1 Integration with state of the art tools

In order to provide to the end-users a natural way of editing and versioning their ontologies, we integrated SemVersion in Protégé (Protégé). The result of this integration is the Versioning Manager Plug-in (Fig.) offering the core functionality described above, but not limited to it. The actual reason behind this integration was to demonstrate the fact that SemVersion is a good solution for enriching a normal ontology editor in order to provide the necessary functionality for completing and maintaining the dynamics in the ontology lifecycle.

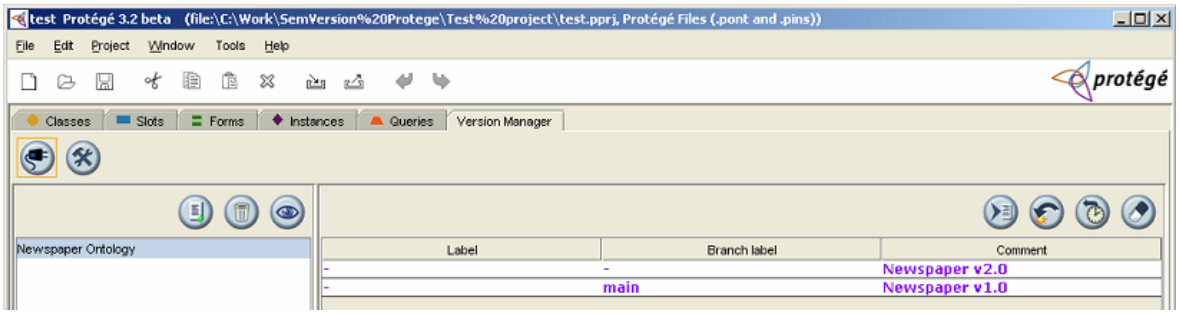

Fig. 7. Version manager integrated in Protégé

In order to support the afore mentioned issues, the Versioning Manager offers the possibility to load a particular version of an ontology for editing as well as to save the current edited ontology as a new version. In terms of visualization, the current status provides structural and semantic diffs visualization of the added and removed statements (tabular format) and also a graphical visualization for the structural diff, but only in terms of classes and subclasses between several versions. Our goal for the diffs visualization, and part of the future developments, is to create an intuitive format, for example, by displaying the two ontologies in parallel and create graphic connections to indicate the added and removed statements.

6. CONCLUSION

Versioning support for ontologies is crucial especially in dynamic environments. We presented in this paper a methodology for RDF-based versioning that separates the management aspects from the versioning core functionality. As functionality, SemVersion provides structural diff as well as semantic diff, considering blank node enrichment as a technique to identify the blank nodes in the versioned models.

SemVersion's architecture is based on RDF2GO and RDFREACTOR. Both choices helped in keeping the programming flexible and fast. Because of them, the triple store can be switched from one to another by

changing a single line of code. This offers a cost effective way to experiment with several triple stores until the best solution for a given use-case is found.

In the future, the biggest challenge will be scalable reasoning, and we are looking forward to upcoming solutions. Until then, SemVersion represents a multi-language versioning system that could help research and industry to employ ontology based technologies in dynamic environments.

ACKNOWLEDGEMENT

Part of this work has been funded by the European Commission 6th Framework Programme in the context of the Knowledge Web - Network of Excellence project, FP6-507482 and the EU IST NEPOMUK IP – The Social Semantic Desktop, FP6-027705.

REFERENCES

Barnejee, J., Kim, W., Kim, H-J., Korth, H. 1987. Semantics and implementation of schema evolution in object-oriented databases. In: *Proc of the Annual Conf on Management of Data (ACM SIGMOD 16(3))*, San Francisco, US

Cobena, G. 2003. *Change management of semi-structured data on the Web*. Phd thesis, INRIA TU-0789.

Huersch, W. 1997. Maintaining consistency and behaviour of object-oriented systems during evolution. In: *Proc of the ACM Conf on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA97)*. pp. 1–21

Klein, M., Fensel, D. 2001. Ontology versioning for the semantic web. In: *1st Int Semantic Web Working Symp. (SWWS)*, Stanford, CA, USA. pp. 75–91

Klein, M., Fensel, D. 2002. OntoView: web-based ontology versioning. *Technical report*, Vrije Universiteit Amsterdam. Submitted, draft at http://www.cs.vu.nl/˜mcaklein/papers/ontoview.pdf

Klein, M. 2004. Change Management for Distributed Ontologies. *Phd thesis*, Vrije Univ. Amsterdam

Kiryakov, A., Simov, K., Ognyanov, D. 2002. Ontology middleware: Analysis and design. *Technical report*, IST Project IST-1999-10132 On-To-Knowledge

Noy, N., Klein, M. 2003. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*

Roddick, J. 1995. A survey of schema versioning issues for database systems. *Information and Software Technology* 37, pp. 383–393

Voelkel, Max, 2005. Writing the Semantic Web with Java. Technical Report. http://www.xam.de/2005/12_voelkel_semweb4j_DERISem05.pdf

Sintek, M., Decker, S. 2002. Triple - a query, inference, and transformation language for the semantic web. In: *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, London, UK, Springer-Verlag. pp. 364–378

Staab, S., Studer, R., eds. 2004. Handbook on Ontologies in Information Systems. *Int Handbooks on Information Systems*. Springer

Stojanovic, L. 2004. Methods and Tools for Ontology Evolution. *PhD Thesis*, University of Karlsruhe.

Su, H., Kramer, D., Rundensteiner, E. 2002. XEM: XML evolution manager. *Technical Report WPI-CSTR-02-09*, Worcester Polytechnic Institute

*, *ISOCO - KPOntology*. http://kpontology.isoco.com/

*, *Protege Ontology Editor*. http://protege.stanford.edu

*, *Web Service Modeling Ontology (WSMO)*. http://www.wsmo.org/

*, *JENA – A Semantic Framework for Java*. http://jena.sourceforge.net/