SVoNt – Version Control of OWL Ontologies on the Concept Level

Markus Luczak-Rösch, Gökhan Coskun, Adrian Paschke, Mario Rothe, Robert Tolksdorf
{luczak,coskun,paschke,mrothe,tolk}@inf.fu-berlin.de

Abstract: Like in Software Engineering the development of ontologies in a team requires efficient support for the management of ontology versions. In this paper we introduce an SVN-based approach for versioning of W3C OWL ontologies called SVoNt – a subversion system for ontologies. Our major goal was to enable views on the revision history of an ontology on the concept level in the same fashion as it is known from the classical SVN approach for the document level.

1 Introduction

Different versions of ontologies can arise in the possibly iterative and collaborative ontology engineering process, as well as during runtime for reusing, extending and refactoring ontologies to incorporate, e.g. knowledge changes and optimizations. Efficient version management is an important functionality in the ontology life cycle management.

Several works on ontology versioning exist which capture especially the key problems of (1) how to detect and track ontology changes, (2) how to merge different ontology versions, and (3) how to release and distribute new versions of ontologies such as the works of Klein[Kle02] and Noy[NM02] amongst others. However, an widely applied engineering tool for ontology version management is still missing.

In this paper we introduce an Apache Subversion (SVN)-based approach for versioning OWL ontologies that conform to the lightweight description logics $\mathcal{EL}$. The system is called SVoNt – subversion for ontologies. Our major goal was to enable a view to the revision history of an ontology on the concept level in the same fashion as it is well-known from the general SVN approach on the file system level. The major difference to other ontology versioning approaches is that we keep the backward compatibility to classical SVN clients. The prototype is implemented as a plug-in for the Eclipse IDE. The adoption and integration into mature ontology development tools which also set up on the Eclipse IDE, such as the NeOn toolkit, TopBraid Composer, and Protégé, is possible in an easy way.

The paper is structured as follows. In the next section we report on existing and related work in the field of ontology versioning. In Section 3, we present the system design of our approach and the current state of the prototype implementation of SVoNt. This paper closes with the conclusions from our recent research and development activities related to SVoNt and a road map of our future steps in Section 4.

2 Related Work

In the past years several approaches to Ontology versioning were pursued. Prompt [NM03] is a collection of tools for ontology versioning realized as a plugin for the Ontology IDE Protégé. The versioning works on a local basis, so this is no classical versioning system. Its feature list contains the calculation of the semantic difference (ontology diff) and the semi-automatical merge of ontologies. Collaborative Protégé [TNTM08] offers another approach to develope ontologies in a group, through an annotation and voting system for changes. Another tool with enabled versioning support is TopBraid Composer[Top] , with usual subversion versioning support and local change history and rollback ability. SemVersion [VG06] is an RDF-centered approach for an ontology versioning system. Model-based versioning is used, so that every version of the ontology is represented by a Triple model. Each version contains Metainformation triple in its model. The Ontology Engineering Platform NeonToolKit (NTK) offers some possibilities for versioning ontologies. The change capturing plugin[NeOa] logs local changes on an ontology and synchronizes them with an external registry. The OWLDiff plugin[NeOb] for NTK enables difference detection and merging of ontologies, two main tasks for a versioning system. Timothy Redmond et al. [RSDT08] shows the design of an ontology version control system that is build up from the scratch. Roman Kontchakov[Kon08] developed a theoretical framework for comparing different versions of DL-Lite ontologies. The problem of the semantic difference of ontologies is handled by an work of Enrico Franconi et al. [EF10].

3 Design and Implementation of SVoNt

The SVoNt system consists of an extended Subversion server and a special SVoNt client. The extensions of the server reuse existing functionalities like logging, authentification and versioning features of Apache Subversion. This allows integrating the SVoNt server into existing Subversion environments, such as Eclipse, and enables classical SVN clients to address the server for OWL document file management - but, without the ontology specific versioning functionalities which can be used only by an extended SVoNt client. In the SVoNt system the W3C Web Ontology Language (OWL) is used as basis for versioning. Each SVN repository represents the evolution of an ontology. To keep the backward compatibility to classical SVN clients we decided that each ontology is located in a specific OWL file in the repository and is not partitioned into several files representing single ontology entities.

To express changes of the structure or the semantics of an ontology an additional versioning mechanism is used, which combines the Subversion system to version the ontology source file and a separate system for concept-based versioning of the ontology. This means that every concept owns a specific revision number, which works equivalent to the Subversion revision system for single files. Therefore conceptual changes of the ontology must be created on the SVoNt server, because Subversion only submits file line deltas to the server. These semantic changes are generated on the server by performing a diff between the updated ontology and the base ontology. This way of generating a difference is not trivial

and only works if the ontologies are syntactically and semantically consistent, thus this consistence has to be checked. The changes are deposited persistently in the conceptual change log and served via an additional external interface used by a SVoNt client.

Because Subversion uses a client-server-architecture and SVoNt directly extends this system, SVoNt is using such an architecture as well. The server stays connectable with a regular SVN client, because the extended SVN interface of the SVoNt system is fully compatible with Subversion. The overall architecture of the SVoNt system is depicted in Figure 1.

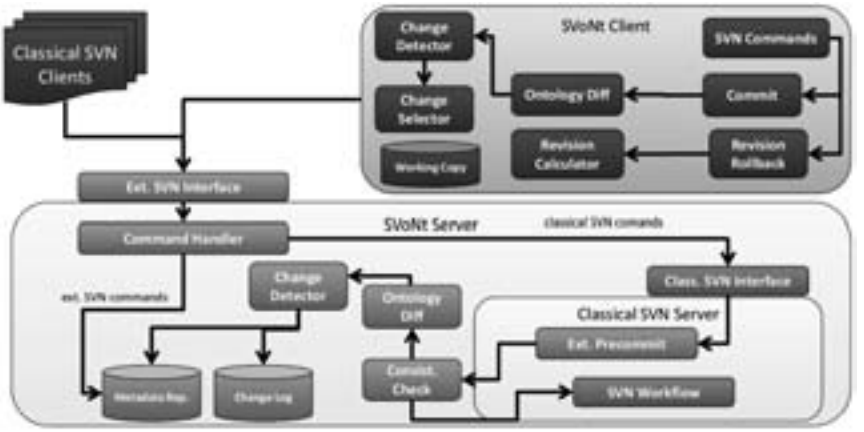

Figure 1: Design of the SVoNt System Architecture

The SVoNt server is an extended SVN server with additional functionality for versioning ontologies. E.g. there is a check done through a precommit-hook to see, if an ontology was changed and needs to be handled in the special ontology versioning process. This process follows a specific procedure running through the modules (1) consitency check, (2) change detection/ontology diff and (3) change log/metadata repository. To allow access to the additional versioning information generated by this process, the access interface of the server was extended. That way an external access apart the classic SVN interface is granted.

As mentioned before the classical SVN clients can use the regular SVN information provided by the SVoNt server. An extended SVoNt client on is a specialized software tailored for versioning ontologies. In addition to the classic SVN revision information it uses ontology-based concept changes from the change log of the SVoNt server. This information is used to display versioning specific metadata in the concept view. A rollback module checks, whether a particular revision of concepts can be undone without generating incompatibilities. Furthermore the client keeps a local change log that stores changes since the last update of the ontology. These, together with the SVoNt server, are detected by a local Change-Detection module. The Change-Selector module allows for the user to choose specific changes to either undo or commit them to the versioning server.

The server-sided extension of Subversion consists of an ontology specific processing of the commit of an ontology, which runs through the three mentioned modules sequentially. On a commit to the SVoNt server a precommit hook is executed on the server to check, if the commit contains an ontology. In such a case the versioning process is triggered. For that the ontology is passed on to an OWL validator, which scans the ontology for specific properties. If that is not the case the execution of the commit is cancelled and the client receives a specific error. In a positive case the Change-Detection module is activated, which generates the difference between the basis ontology and the updated ontology. If no changes between the ontologies are detected, the precommit is succeeded and the ontology is being versioned in Subversion. This is possible if textual comments in the ontology or the order of concepts are changed, leaving the semantics of the ontology untouched. In the case of detecting changes with the ontology diff, these changes are passed to the Change log module, which writes them to the change log of that revision. With that the ontology specific processing is completed and the ontology can be added to the SVN repository and being versioned.

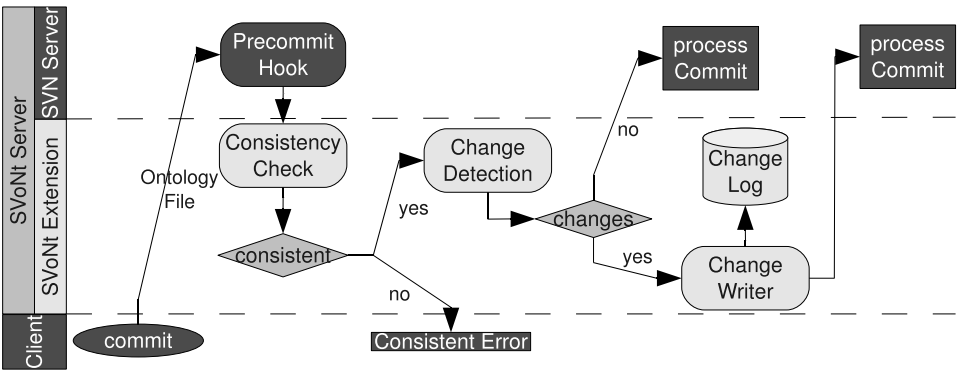

Figure 2: Workflow of the SVoNt Commit Operation

The connection between the additional SVoNt specific components to the SVN server is done via precommit hook scripts provided by Subversion, what enables to interfere with the commit process. The SVoNtRunner, whose `run()` method triggers the ontology specific versioning process, is the heart of the implementation. For the standard implementation the validation module the OWL reasoner Pellet with the OWL-API is used to check the syntactic and semantic validity of the ontology.

The Change-Detection (Diff) module uses algorithms of the OWLdiff (*http://krizik.felk.cvut.cz/km/owldiff/*) library, which can be integrated in adapted form. It provides (1) the basic ontology comparison algorithm and (2) the CEX logical diff. The first algorithm is an implementation of an axiomatic difference generation, which detects the structural changes between the ontologies and returns those axioms that were added or removed. With an entailment checking with Pellet it is possible to determine the semantic equivalence of both ontologies. The OWLdiff implementation of the CEX Logical Diff Algorithm is based on the work of Konev et al.[KWW08] and is able to detect pervasive changes

in the semantics of two ontologies which comply to the $\mathcal{EL}$ description logics. The algorithm returns two sets of concepts which represent the difference between two ontologies which cannot be detected in the class hierarchies. In detail this solution bases on deciding the Σ-entailment for the lightweight description logics $\mathcal{EL}$ which means that it decides whether two compared terminologies imply the same concept implications with reference to a common signature. This implementation is used to identify concepts as being changed in marking all concepts as semantically changed, which are returned by CEX-Diff. Ontologies besides the $\mathcal{EL}$ language profile have to use the basic diff algorithm. The semantic changes of the ontology being recognized are brought into a persistent state by generating a file in the Ontology Metadata Vocabulary (OMV) format for each commit, which represents a revision.

4 Conclusions and Future Work

In this paper we presented the SVN-based ontology versioning system SVoNt. Our next step is to run a case study with some real life data of one of the industrial partners of the German research project Corporate Semantic Web[1] – the Ontonym GmbH – which is a provider of ontology-based services which supply the appropriate background knowledge for the client's applications and services. Ontonym needs an efficient ontology versioning strategy since the company has to handle several often and rapidly changing ontologies which are accessed and maintained by several people internally. On the one hand, these people are the ontology engineers but on the other hand software engineers have to access the ontology as well from time to time. The former prune and refine the ontology depending on new knowledge they observed by evaluating queries and responses of their services as well as time consuming research in the customer's domain. The latter have to align the service functionalities depending on the capabilities of the ontology. In summary, Ontonym needs an ontology version control system for users with disparate viewpoints and skills. SVoNt is designed to fulfill this requirement because it adopts principles from the well-known SVN system and may be used with classical SVN clients.

We also started to implement and study some simple graph analyses which allow representing the content and the evolution of the ontologies through appropriate visualization techniques and envision to simplify comprehensibility. To achieve that, we investigated in ontology visualization techniques (based upon the SONIVIS:Tool (*www.sonivis.org*)) which simplify the reuse of ontologies by reducing the costs for analyzing ontologies.

In future work, we will extend these visualization techniques with versioning information from SVoNt, so that information about the change between different versions can be visualized. By combining the information about the structure of the ontology and its evolution it is possible to identify frequently changing parts within the ontology and to mark them as unreliable and unstable sections that need further attention. On the other hand, parts of the ontology model, which do not change over several versions can be marked as stable and reliable.

[1] *www.corporate-semantic-web.de*

Acknowledgement

This work has been partially supported by the "InnoProfile-Corporate Semantic Web" project funded by the German Federal Ministry of Education and Research (BMBF).

References

[EF10] I. Varzinczak E. Franconi, T. Meyer. Semantic Diff as the Basis for Knowledge Base Versioning. In *13th Workshop on Non-Monotonic Reasoning (NMR)*, 2010.

[Kle02] Michel A. C. Klein. Versioning of Distributed Ontologies. Technical Report Del 20, Vrije Universiteit Amsterdam, december 2002.

[Kon08] Roman Kontchakov. Can you tell the difference between DL-Lite ontologies. In *In Proceedings of KR'08*, pages 285–295. AAAI Press, 2008.

[KWW08] Boris Konev, Dirk Walther, and Frank Wolter. The Logical Difference Problem for Description Logic Terminologies. In *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2008.

[NeOa] NeOn Toolkit. Change Capturing Plugin. http://www.neon-toolkit.org/wiki/Change_Capturing. Visited on June 17th, 2010.

[NeOb] NeOn Toolkit. OWLDiff Plugin. http://www.neon-toolkit.org/wiki/OWLDiff. Visited on June 17th, 2010.

[NM02] Natalya F. Noy and Mark A. Musen. PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions. In *National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 744–750, Edmonton, Alberta, Canada, July 2002.

[NM03] Natalya F. Noy and Mark A. Musen. The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[RSDT08] Timothy Redmond, Michael Smith, Nick Drummond, and Tania Tudorache. Managing Change: An Ontology Version Control System. In *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[TNTM08] Tania Tudorache, Natalya Fridman Noy, Samson Tu, and Mark A. Musen. Supporting Collaborative Ontology Development in Protégé. In *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2008.

[Top] TopQuadrant. TopBraid Composer. http://www.topquadrant.com/products/TB_Composer.html. Visited on June 17th, 2010.

[VG06] Max Völkel and Tudor Groza. SemVersion: RDF-based Ontology Versioning System. In *Proceedings of the IADIS International Conference WWW / Internet 2006 (ICWI 2006)*, 2006.