ContentCVS: A CVS-based Collaborative ONTology ENgineering Tool

E. Jiménez-Ruiz[1]*, B. Cuenca Grau[2]**, I. Horrocks[2], and R. Berlanga[1]

[1] Universitat Jaume I, Spain
[2] Oxford University Computing Laboratory, UK,

Abstract. We present ContentCVS, a system that implements a novel approach to facilitate the collaborative development of ontologies. Our approach adapts Concurrent Versioning, a successful paradigm in collaborative software development, to allow several developers to make changes concurrently to an ontology. Conflict detection and resolution are based on novel techniques that take into account the structure and semantics of the ontology versions to be reconciled by using precisely-defined notions of structural and semantic differences between ontologies and by extending existing ontology debugging and repair techniques.

1 Motivation

OWL Ontologies are already being used in many application domains. In particular, OWL is extensively used in life sciences; prominent examples of OWL ontologies are the National Cancer Institute (NCI) Thesaurus, SNOMED CT, the Gene Ontology (GO), the Foundational Model of Anatomy (FMA), and GALEN.

Most realistic ontologies, including the ones just mentioned, are being developed collaboratively. Maintaining such large ontologies in a collaborative way is a highly complex process: developers need to regularly merge and reconcile their modifications to ensure that the ontology captures a consistent unified view of the domain. Changes performed by different users may, however, conflict in complex ways and lead to errors. These errors may manifest themselves both as structural (i.e., syntactic) mismatches between developers' ontological descriptions, and as unintended logical consequences.

Tools supporting collaboration should therefore provide means for: *(i)* keeping track of ontology versions and changes and reverting, if necessary, to a previously agreed upon version, *(ii)* comparing potentially conflicting versions and identifying conflicting parts, *(iii)* identifying errors in the reconciled ontology constructed from conflicting versions, and *(iv)* suggesting possible ways to repair the identified errors with a minimal impact on the ontology.

In this paper we present ContentCVS[3], a system that is available for download as a Protégé 4 plugin [1]. In sections 2-5 we briefly describe the functionality provided by ContentCVS to satisfy each of the above-mentioned requirements. Finally, in Section 6 we summarize the results of a pilot user study that we have conducted to evaluate the usability of the system. For additional information, we refer the reader to [1].

* The author is supported by the PhD Fellowship of the *Generalitat Valenciana*.
** The author is supported by a Royal Society University Research Fellowship.

[3] A **C**ollaborative **ONT**ology **EN**gineering **T**ool.

2 Keeping Track of Versions and Changes

ContentCVS closely follows the *Concurrent Versioning* paradigm. The change history and the most recent version $\mathcal{O}^R$ of the ontology, which represents the developers' shared understanding of the domain, are kept in a server's shared repository. Each developer with access to the repository can connect to the server to *check out* a copy of $\mathcal{O}^R$, allowing developers to work on their own local copy $\mathcal{O}^L$, which they can modify at will. Developers can *commit* their changes to the server at any time. This allows several developers to make changes concurrently to the shared ontology. To keep the system in a consistent state, the server only accepts changes to the latest version of the shared ontology. Developers should hence use the CVS client to regularly *update* their local copy with changes made by others.

Manual intervention is only needed when a *conflict* arises between a committed version in the server and a yet-uncommitted local version. Conflicts are reported whenever the two compared versions of an ontology are not "equivalent" according to a given notion of equivalence, which we describe next.

3 Detecting Conflicts

A typical CVS system treats the files in a software project as ordinary text files and hence checking equivalence amounts to determining whether the two compared files are *syntactically equal* (i.e., they contain exactly the same characters in exactly the same order). This notion of equivalence is, however, too strict in the case of ontologies, since OWL files have very specific structure and semantics. For example, if two OWL files are identical except for the fact that the axioms appear in different order, the corresponding ontologies should be clearly treated as equivalent: an ontology contains a set of axioms and hence the order in which they appear is irrelevant.

An alternative in the case of ontologies is to use the notion of *logical equivalence*. This notion is, however, too permissive: even if two ontologies are logically equivalent (the strongest assumption from a semantic point of view), conflicts may still exist. This might result from incompatible annotations (statements that act as comments and do not carry logical meaning), mismatches in modelling styles, and so on.

ContentCVS uses the notion of *structural equivalence* between OWL 2 ontologies [2] to compare ontology versions and identify conflicts. Intuitively, this notion is based solely on comparing the OWL modeling structures. For example, changes in the order in which structures such as axioms and conjunction of concepts appear in the ontology file are irrelevant. Structural equivalence is thus more permissive than syntactical equality, but stricter than logical equivalence. Figure 1 shows the structural comparison between two ontologies in ContentCVS. The left-hand-side shows axioms in $\mathcal{O}^L$ that do not have an structurally equivalent axiom in $\mathcal{O}^R$; the right-hand-side shows the axioms in $\mathcal{O}^R$ without an structurally equivalent axiom in $\mathcal{O}^L$. All the axioms in Figure 1 are thus regarded as conflicts. To facilitate the comparison, the conflicting axioms are sorted and aligned according to the entities they define. ContentCVS also provides functionality for examining and comparing the change histories of the local and repository ontologies since the latest CVS update operation.

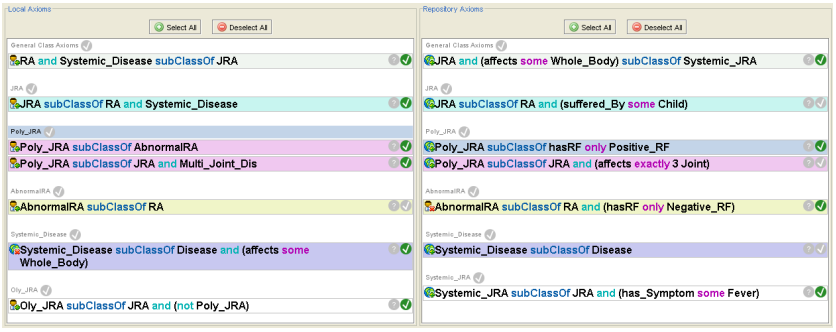


Fig. 1. GUI for Displaying Structural Differences in ContentCVS

4 Building a Reconciled Ontology and Identifying Errors

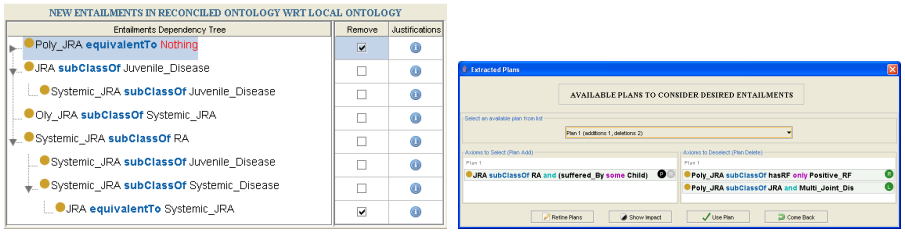
Once the user has selected the desired axioms from the structural difference (marked with a ✓ in Figure 1) a temporary version of the reconciled ontology $\mathcal{O}^L_{\mathsf{tmp}}$ is built from the non-conflicting part of $\mathcal{O}^L$ plus the axioms selected from the structural difference. At this point, the user may declare the conflict resolved and commit $\mathcal{O}^L_{\mathsf{tmp}}$ to the repository, in which case conflict resolution remains a purely syntactic process. Otherwise, ContentCVS allows the ontology developer to use a reasoner to examine the semantic consequences of their choices and ensure that $\mathcal{O}^L_{\mathsf{tmp}}$ meets their requirements (typically, that it includes as much intended information as possible without leading to inconsistencies or other undesired entailments).

To facilitate error detection, ContentCVS compares the entailments that hold in $\mathcal{O}^L_{\mathsf{tmp}}$ with those that hold in $\mathcal{O}^L$ and $\mathcal{O}^R$ by using the notion of *logical difference* [3]. Roughly, the logical difference between $\mathcal{O}$ and $\mathcal{O}'$ is the set of all axioms entailed by $\mathcal{O}$, but not by $\mathcal{O}'$. This difference can in theory be infinite, and ContentCVS allows users to (finitely) approximate it by indicating which types of entailments should be included. For example, users can ask ContentCVS to compute differences in the entailed concept and role hierarchies, differences in the entailed disjointness axioms, differences in entailed axioms of the form $A \sqsubseteq \exists R.B$ (with A and B atomic), and so on.

Figure 2(a) shows the GUI in ContentCVS for displaying *new entailments*, that is, those that hold in $\mathcal{O}^L_{\mathsf{tmp}}$ but not in $\mathcal{O}^L$. The GUI allows users to indicate which of those entailments are unintended and should be "deleted". ContentCVS provides a similar GUI for displaying *lost entailments* (those that hold in $\mathcal{O}^L$ but not in $\mathcal{O}^L_{\mathsf{tmp}}$) and for selecting which of those should be "recovered".

5 Repairing Errors in the Reconciled Ontology

Changing the set of entailments in an ontology can only be achieved by modifying the ontology itself (see for example [4]). Hence, if the user selects any new entailments to "delete" or any lost entailments to "recover", then $\mathcal{O}^L_{\mathsf{tmp}}$ clearly needs to be modified.



(a) Entailment Selection in ContentCVS (b) Plan Selection in ContentCVS

Fig. 2. Error Detection and Plan Visualization in ContentCVS

ContentCVS provides functionality for automatically suggesting the required modifications. Each possible solution (called a *repair plan*) is presented to the user as a set of axioms to be removed from $\mathcal{O}^L_{\text{tmp}}$ and a set of axioms to be added to $\mathcal{O}^L_{\text{tmp}}$. ContentCVS implements a number of heuristics for ranking the possible repair plans and provides a GUI (see Figure 2(b)) that indicates whether the axioms in the plan come from $\mathcal{O}^L$ or $\mathcal{O}^R$ (marked with "L", "R" respectively), and whether an axiom is shared by all plans (marked with a "P").

6 Conclusions from User Study

We conducted a pilot user study to evaluate the usability of ContentCVS and to show the adequacy of our approach in practice (see [1] for details). Our results show that most users considered very useful the implemented CVS functionality as well as the computation of structural differences between ontology versions. Users were also satisfied with the detection of errors using logical differences: when using ContentCVS all the participants could identify both new unintended entailments and lost intended entailments. Most users were also satisfied with the functionality for computing repair plans as well as with the reconciled ontology finally obtained.

The main points of criticism were, on the one hand, the excessive amount of information displayed when using "large" approximations of the logical difference and, on the other hand, slow response of the tool when performing reasoning-intensive tasks. We consider addressing these deficiencies as a part of our future work.

References

[1] Jimenez-Ruiz, E., Cuenca Grau, B., Horrocks, I., R.Berlanga: Conflict detection and resolution in collaborative ontology development. Technical report (2009) , Tool and user study available at: http://krono.act.uji.es/people/Ernesto/contentcvs.

[2] Motik, B., Patel-Schneider, P., B.Parsia: OWL 2 structural specification and functional-style syntax. W3C Recommendation (2009) . Available at http://www.w3.org/TR/owl2-syntax/.

[3] Konev, B., Walther, D., Wolter, F.: The logical difference problem for description logic terminologies. In: Proc. of IJCAR. (2008) 259–274

[4] Kalyanpur, A., Parsia, B., Sirin, E., Cuenca Grau, B.: Repairing unsatisfiable concepts in OWL ontologies. In: Proc. of ESWC. (2006) 170–184