# Building Ontologies Collaboratively Using ContentCVS

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

[1] Universitat Jaume I, Spain, {ejimenez,berlanga}@uji.es
[2] University of Oxford, UK, {berg,ian.horrocks}@comlab.ox.ac.uk

## 1 Motivation

OWL Ontologies are already being used in many application domains. In particular, OWL is extensively used in the clinical 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.

These ontologies are large and complex; for example, SNOMED currently describes more than $350.000$ concepts whereas NCI and GALEN describe around $50.000$ concepts. Furthermore, these ontologies are in continuous evolution; for example the developers of NCI and GO perform approximately $350$ additions of new entities and $25$ deletions of obsolete entities each month [1].

Most realistic ontologies, including the ones just mentioned, are being developed collaboratively. The developers of an ontology can be geographically distributed and may contribute in different ways and to different extents. Maintaining such large ontologies in a collaborative way is a highly complex process, which involves tracking and managing the frequent changes to the ontology, reconciling conflicting views of the domain from different developers, minimising the introduction of errors (e.g., ensuring that the ontology does not have unintended logical consequences), and so on.

In this setting, 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 order to address *(i)*, we propose to adapt the Concurrent Versioning— a successful paradigm in collaborative software development— to allow several developers to make changes concurrently to the same ontology (see Section 2). To address *(ii)* we propose a notion of *conflict* between ontology versions and provide means for identifying conflicting parts based on it (see Section 3). To address *(iii)* we propose in Section

* This work was partially funded by the PhD Fellowship program of the *Generalitat Valenciana* and the Spanish National Research Program, contract number TIN2008-01825/TIN.

4 a framework for comparing the entailments that hold in the compared versions and in the reconciled ontology, based on the notion of a *deductive difference* [2, 3] and also describe techniques for helping users decide which of the reported entailments are intended. Finally, to address *(iv)*, we propose in Section 4 several improvements to existing techniques for ontology debugging and repair [4, 5, 6, 7] and adapt them to our setting. In Sections 2, 3, and 4, we describe both our general approach and algorithmic techniques as well as their implementation in our tool ContentCVS,[3] a Protégé 4 plugin freely available for download.[4] We finally add an empirical evaluation showing that our approach is useful and feasible in practice. Due to lack of space, certain details have been included in an extended version of this paper, which is available online [8].

## 2 CVS-based collaboration

In software engineering, the Concurrent Versioning paradigm has been very successful for collaboration in large projects. A Concurrent Versioning System (CVS) uses a client-server architecture: a CVS server stores the current version of a project and its change history; CVS clients connect to the server to create (*export*) a new repository, *check out* a copy of the project, allowing developers to work on their own 'local' copy, and then later to *commit* their changes to the server. This allows several developers to make changes concurrently to a project. To keep the system in a consistent state, the server only accepts changes to the latest version of any given project file. Developers should hence use the CVS client to regularly commit their changes and *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 a file are not *equivalent* according to a given notion of equivalence between versions of a file.

Our tool ContentCVS closely follows this paradigm. The most recent version $\mathcal{O}^R$ of the ontology, which represents the developers' shared understanding of the domain, is kept in a server's shared repository. Each developer with access to the repository maintains a local copy $\mathcal{O}^L$, which can be modified at will. This local copy can be either in conflict with $\mathcal{O}^R$ ($\mathsf{conflict} = \mathsf{true}$) or not in conflict ($\mathsf{conflict} = \mathsf{false}$). The system also maintains version numbers $v_R$ and $v_L$ for $\mathcal{O}^R$ and $\mathcal{O}^L$ respectively as well as a local 'backup' copy $\mathcal{O}^L_{\mathsf{bak}}$ of the latest local version that was 'written' to the repository. At any time, a developer can access the repository using the basic operations *export*, *check-out*, *update* and *commit*. In ContentCVS, these operations often involve checking whether two ontology files $\mathcal{O}$ and $\mathcal{O}'$ are 'equivalent' under a specific notion of equivalence between ontology files which will be introduced in Section 3 (denoted $\mathcal{O} \sim \mathcal{O}'$).

As an illustration, Figure 1 provides the semantics of the commit operation in ContentCVS. For the semantics of the other CVS operations we refer the reader to [8]. In Figure 1, note that if $\mathcal{O}^L_{\mathsf{bak}} \backsim \mathcal{O}^L$ then there are no meaningful local changes and no action is required. Otherwise, the commit process only succeeds if $\mathcal{O}^L$ is up-to-date ($v_L = v_R$) and not in conflict ($\mathsf{conflict} = \mathsf{false}$). In case of success, the commit involves replacing $\mathcal{O}^R$ with $\mathcal{O}^L$ and incrementing the version number.

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

[4] ContentCVS: `http://krono.act.uji.es/people/Ernesto/contentcvs`

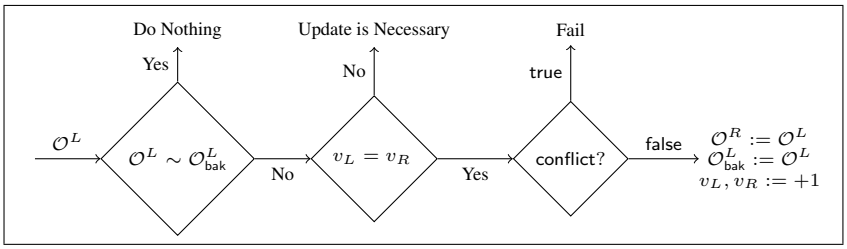



**Fig. 1.** Semantics of the commit operation in ContentCVS

## 3 Change and conflict detection

As mentioned in Section 2, change or conflict detection amounts to checking whether two compared versions of a file are not 'equivalent' according to a given notion of equivalence between versions of a file.

A typical CVS treats the files in a software project as 'ordinary' text files and hence checking equivalence amounts to determining whether the two versions 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 two axioms appear in different order, the corresponding ontologies should be clearly treated as 'equivalent': an ontology contains a *set* of axioms and hence their order is irrelevant [9].

Another possibility is to use the notion of logical equivalence. This notion of equivalence is, however, too permissive: even if $\mathcal{O} \equiv \mathcal{O}'$—the strongest assumption from a semantic point of view—conflicts may still exist. This might result from the presence of incompatible annotations (statements that act as comments and do not carry logical meaning) [9], or a mismatch in modelling styles; for example, $\mathcal{O}$ may be written in a simple language such as the OWL 2 EL profile [9, 10] and contain $\alpha := (A \sqsubseteq B \sqcap C)$, while $\mathcal{O}'$ may contain $\beta := (\neg B \sqcup \neg C \sqsubseteq \neg A)$. Even though $\alpha \equiv \beta$, the explicit use of negation and disjunction means that $\mathcal{O}'$ is outside the EL profile.

Therefore, the notion of a conflict should be based on a notion of ontology equivalence 'in-between' syntactical equality and logical equivalence. We propose to borrow the notion of *structural equivalence* from the OWL 2 specification [11]. Intuitively, this notion of equivalence is based solely on comparing structures by using the definition of the modeling constructs available in OWL and OWL 2; for example, several modeling constructs are defined as sets of objects (e.g., ontologies are defined as sets of axioms, conjunction of concepts as a set of conjuncts, and so on); hence changes in the order in which these set elements appear in the ontology file should be seen as irrelevant. For a definition of structural equivalence in DL syntax, we refer the reader to [8].

The use of the notion of structural equivalence as a formal basis for detecting conflicts between ontology versions presents a number of compelling advantages. First, it rules out irrelevant syntactic mismatches based solely on the structure of the OWL language. Second, structurally equivalent ontologies are also logically equivalent. Third, it preserves species and profiles [10] of OWL and OWL 2 respectively; for example

if $\mathcal{O}$ and $\mathcal{O}'$ are structurally equivalent and $\mathcal{O}$ is in OWL Lite (respectively in any of the profiles of OWL 2), then $\mathcal{O}'$ is also in OWL Lite (respectively in the same OWL 2 profile as $\mathcal{O}$). Fourth, it takes into account extra-logical components of an ontology, such as annotations. Finally, it is an agreed-upon notion, defined within the W3C OWL Working Group during the standardisation of OWL 2 and therefore it is supported by mainstream ontology development APIs, such as the OWL API.

Finally, the identification of the conflicting parts in $\mathcal{O}$ and $\mathcal{O}'$ using the notion of structural equivalence can be performed by computing their *structural difference*.

**Definition 1.** *The structural difference between $\mathcal{O}_1$ and $\mathcal{O}_2$ is the set $\Lambda_s$ of axioms $\alpha \in \mathcal{O}_i$ for which there is no $\beta \in \mathcal{O}_j$ s.t. $\alpha$ and $\beta$ are structurally equivalent with $i, j \in \{1, 2\}$ and $i \neq j$.*

## 4 Conflict resolution

Conflict resolution is the process of constructing a reconciled ontology from two ontology versions which are in conflict. In a CVS, the conflict resolution functionality is provided by the CVS client. Our approach is based on the principle that a CVS client should allow users to resolve the identified conflicts at two different levels: *(i) structural*, where only the structure of the compared ontology versions is taken into account to build the reconciled ontology (see Section 4.1); *(ii) structural and semantic*, where both the structure and the logical consequences of the compared ontology versions as well as of the reconciled ontology are considered (see Sections 4.1—4.4).

Our approach is summarised in Table 1. The steps marked with $(\checkmark)$ require human intervention. We next describe in detail each of the steps in Table 1. To illustrate our techniques, we use as a running example the collaborative development of an ontology about arthritis, which is a real use case that we have encountered within the Health-e-Child (HeC) project.[5] We consider the case were two developers, John and Anna, independently extend a version of the arthritis ontology by describing types of systemic arthritis and juvenile arthritis respectively. Both John and Anna define a kind of arthritis called JRA (Juvenile Rheumatoid Arthritis). Hence, even if largely distinct, the domains described by John and Anna overlap, which leads to conflicts.

### 4.1 Selection of axioms using structural difference

Conflict resolution in text files is usually performed by first identifying and displaying the conflicting sections in the two files (e.g., a line, or a paragraph) and then manually select the desired content. Analogously, our proposal for structural conflict resolution involves the selection of the conflicting axioms $\mathcal{S}$ (i.e., those in the structural difference) to be included in a (provisional) version $\mathcal{O}^L_{\mathsf{temp}}$ of $\mathcal{O}^L$ (Step 1). The ontology $\mathcal{O}^L_{\mathsf{temp}}$ is obtained from the non-conflicting part of $\mathcal{O}^L$ plus the selected axioms $\mathcal{S}$ (Step 2).

After constructing $\mathcal{O}^L_{\mathsf{temp}}$, the user may declare the conflict resolved (Step 3), in which case conflict resolution remains a purely syntactic process—as in the case of text files. Otherwise, ontology developers can use a reasoner to examine the semantic consequences of their choices and make sure that $\mathcal{O}^L_{\mathsf{temp}}$ is error-free.

[5] Health-e-Child project: `http://www.health-e-child.org`

**Input:** $\mathcal{O}^L, \mathcal{O}^R$: ontologies with $\mathcal{O}^L \not\asymp \mathcal{O}^R$, conflict $=$ true and structural difference $\Lambda_s$
**Output:** $\mathcal{O}^L$: ontology; conflict: boolean value;

1: (✓) Select $\mathcal{S} \subseteq \Lambda_s$
2: $\mathcal{O}^L_{\text{temp}} := (\mathcal{O}^L \setminus \Lambda_s) \cup \mathcal{S}$
3: (✓) **if** $\mathcal{O}^L_{\text{temp}}$ is satisfactory **return** $\mathcal{O}^L := \mathcal{O}^L_{\text{temp}}$, conflict $:=$ false
4: (✓) Select approximation function appr
5: Compute $\text{diff}_{\approx}(\mathcal{O}^L_{\text{temp}}, \mathcal{O}^L)$, $\text{diff}_{\approx}(\mathcal{O}^L_{\text{temp}}, \mathcal{O}^R)$ $\text{diff}_{\approx}(\mathcal{O}^L, \mathcal{O}^L_{\text{temp}})$ and $\text{diff}_{\approx}(\mathcal{O}^R, \mathcal{O}^L_{\text{temp}})$
6: (✓) Select $\Im^+ \subseteq \text{diff}_{\approx}(\mathcal{O}^L_{\text{temp}}, \mathcal{O}^L) \cup \text{diff}_{\approx}(\mathcal{O}^L_{\text{temp}}, \mathcal{O}^R)$
7: (✓) Select $\Im^- \subseteq \text{diff}_{\approx}(\mathcal{O}^L, \mathcal{O}^L_{\text{temp}}) \cup \text{diff}_{\approx}(\mathcal{O}^R, \mathcal{O}^L_{\text{temp}})$
8: Compute minimal plans $\mathbb{P}$ for $\mathcal{O}^L_{\text{temp}}$ given $\Im^+$, $\Im^-$, $\mathcal{O}^+ := \Lambda_s \setminus \mathcal{S}$, and $\mathcal{O}^- := \mathcal{S}$
9: (✓) **if** no satisfactory plan in $\mathbb{P}$, **return** $\mathcal{O}^L$, conflict $:=$ true
10: (✓) Select $\mathcal{P} = \langle \mathcal{P}^+, \mathcal{P}^- \rangle \in \mathbb{P}$
11: **return** $\mathcal{O}^L := (\mathcal{O}^L_{\text{temp}} \cup \mathcal{P}^+) \setminus \mathcal{P}^-$, conflict $:=$ false

**Table 1.** Conflict Resolution Method.

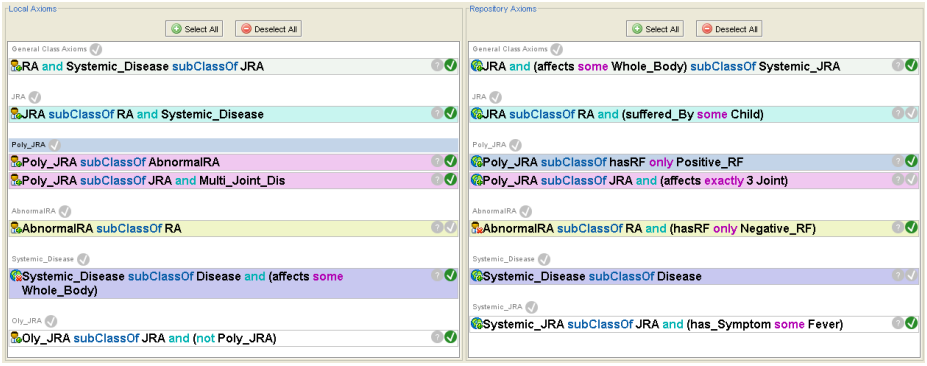



**Fig. 2.** GUI for Structural Differences in ContentCVS

ContentCVS implements a simple GUI to facilitate the selection of axioms from the structural difference, which is shown in Figure 2 for our running example, where John's and Anna's versions are respectively shown on the left-hand-side and on the right-hand-side of the figure. The left-hand-side (respectively the right-hand-side) of the figure shows the axioms in $\Lambda_s \cap \mathcal{O}^L$ (respectively in $\Lambda_s \cap \mathcal{O}^R$).

To facilitate the comparison, axioms are sorted and aligned according to the entities they define. Axioms not aligned with others are shown last in a distinguished position. The selected axioms are indicated in the GUI using a highlighted (✓). Furthermore, ContentCVS provides additional functionality for determining the origin of each axiom in the structural difference. In particular, ContentCVS, indicates whether an axiom appears in the difference as a result of an addition or a deletion by comparing $\mathcal{O}^L$ and $\mathcal{O}^R$ to the local 'backup' copy $\mathcal{O}^L_{\text{bak}}$ of the latest local version that was 'written' to the repository. For example, the axiom $(\text{Poly\_JRA} \sqsubseteq \text{AbnormalRA})$ on the left-hand-side of Figure 2 was added to $\mathcal{O}^L_{\text{bak}}$ in the local ontology (see icon representing a user with a '+'), whereas the axiom $(\text{Systemic\_Disease} \sqsubseteq \text{Disease} \sqcap \exists \text{affects.Whole\_Body})$ was deleted in the repository (see icon representing the Globe with a '×').

### 4.2 Deductive differences

Errors in the reconciliation process can be detected using a reasoner, by computing the logical consequences of the reconciled ontology $\mathcal{O}^L_{\mathsf{temp}}$ and comparing them to those of $\mathcal{O}^L$ and $\mathcal{O}^R$. Errors in $\mathcal{O}^L_{\mathsf{temp}}$ could manifest themselves, however, not only as unsatisfiable concepts or unintended (or missing) subsumptions between atomic concepts, but also as unintended (or missing) entailments involving complex concepts. To help users detect such errors, we propose to compare the entailments that hold in $\mathcal{O}^L_{\mathsf{temp}}$ with those that hold in $\mathcal{O}^L$ and $\mathcal{O}^R$ by using the notion of *deductive difference* [2, 3].

**Definition 2.** *The deductive difference* $\mathsf{diff}(\mathcal{O}, \mathcal{O}')$ *between* $\mathcal{O}$ *and* $\mathcal{O}'$ *expressed in a DL* $\mathcal{DL}$ *is the set of* $\mathcal{DL}$*-axioms* $\alpha$ *s.t.* $\mathcal{O} \not\models \alpha$ *and* $\mathcal{O}' \models \alpha$.

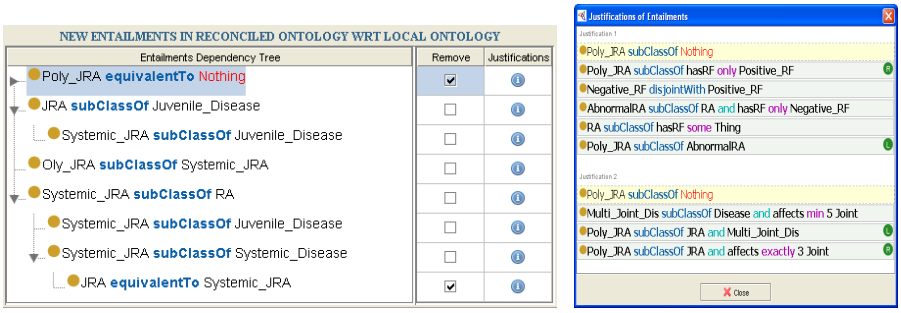
Intuitively, this difference is the set of *all* (possibly complex) entailments that hold in one ontology but not in the other. These may be entailments that *(i)* hold in $\mathcal{O}^L_{\mathsf{temp}}$ and not in $\mathcal{O}^L$; *(ii)* hold in $\mathcal{O}^L_{\mathsf{temp}}$ but not in either $\mathcal{O}^L$ or $\mathcal{O}^R$; *(iii)* hold in $\mathcal{O}^L$ but not in $\mathcal{O}^L_{\mathsf{temp}}$, and *(iv)* hold in $\mathcal{O}^R$ but not in $\mathcal{O}^L_{\mathsf{temp}}$. Therefore, we argue that the relevant deductive differences between $\mathcal{O}^L_{\mathsf{temp}}$, $\mathcal{O}^L$ and $\mathcal{O}^R$ capture all potential errors that may have been introduced in the reconciliation process.

Considering complex entailments obviously comes at a price, both in terms of computational cost and of complication of the GUI. In particular, checking whether $\mathsf{diff}(\mathcal{O}, \mathcal{O}') = \emptyset$ is undecidable in expressive DLs [2]. Furthermore, the number of entailments in the difference can be large (even infinite), and so likely to overwhelm users. These practical drawbacks motivate the need for *approximations*— subsets of the deductive difference (see Step 4 in Table 1).

**Definition 3.** *A function* $\mathsf{diff}_{\approx}(\mathcal{O}, \mathcal{O}')$ *is an approximation for* $\mathsf{diff}(\mathcal{O}, \mathcal{O}')$ *if for each pair* $\mathcal{O}, \mathcal{O}'$ *of* $\mathcal{DL}$*-ontologies,* $\mathsf{diff}_{\approx}(\mathcal{O}, \mathcal{O}') \subseteq \mathsf{diff}(\mathcal{O}, \mathcal{O}')$.

A useful approximation should be easy to compute, yet still provide meaningful information to the user. One possibility is to define an approximation by considering only entailments of a certain form. Our tool ContentCVS allows users to customise approximations by selecting among the following kinds of entailments, where $A, B$ are atomic concepts (including $\top$, $\bot$) and $R, S$ atomic roles or inverses of atomic roles: (i) $A \sqsubseteq B$, (ii) $A \sqsubseteq \neg B$, (iii) $A \sqsubseteq \exists R.B$, (iv) $A \sqsubseteq \forall R.B$, and (v) $R \sqsubseteq S$. The smallest implemented approximation considers only axioms of the form (i), which amounts to comparing the classification hierarchy of both ontologies, while the largest considers all types (i)—(v). Clearly, the larger the class of entailments presented to the user, the more errors could be detected. The corresponding differences, however, are harder to compute, harder to present to the user, and may be harder for the user to understand.

Although these approximations can be algorithmically computed, only the entailments of the form (i) and (v) are typically obtained as standard output of classification. Computing approximations based on entailments (ii)-(iv) can be expensive since it may involve performing a huge amount of entailment tests. To reduce the number of tests, ContentCVS uses the notion of a *locality-based module* [12]. When checking for all entailments of the form (ii)-(iv), ContentCVS first extracts the locality-based module for $A$ and looks for entailments only within the module, which significantly reduces the search space and makes the computation of these approximations practically feasible.




(a) Selection of Entailments (b) Justifications Poly_JRA $\sqsubseteq \bot$

**Fig. 3.** GUI for Selection of Entailments in ContentCVS

### 4.3 Selection of entailments

While some entailments in the computed differences are intended, others reveal errors in $\mathcal{O}^L_{\mathsf{temp}}$. Steps 6 and 7 thus involve selecting entailments that: *(i)* are intended and should follow from $\mathcal{O}^L_{\mathsf{temp}}$ (written $\Im^+$ in Table 1), and *(ii)* are unintended and should not follow from $\mathcal{O}^L_{\mathsf{temp}}$ (written $\Im^-$).

The development of techniques to help users understand the relevant deductive differences and subsequently select the sets of intended and unintended entailments is especially challenging. First, a tool should explain why, on the one hand, the new entailments that hold in $\mathcal{O}^L_{\mathsf{temp}}$ do not hold in $\mathcal{O}^L$ and $\mathcal{O}^R$ alone and, on the other hand, the lost entailments that hold in $\mathcal{O}^L$ and $\mathcal{O}^R$ do not hold in $\mathcal{O}^L_{\mathsf{temp}}$. The notion of a *justification*—that is, a minimal subset of the ontology for which the given entailment holds— has proved very useful in ontology debugging [5, 6]. In order to explain a given entailment, ContentCVS presents all its justifications. Computing all justifications is expensive, so ContentCVS uses the infrastructure and optimisations already available in Protégé [7] and implements those in [13].

Second, the potentially large number of relevant entailments may overwhelm users. These entailments should be presented in a way that makes them easier to understand and manage. ContentCVS extends known ontology debugging techniques by identifying dependencies between entailments. Intuitively, an entailment $\beta$ *depends* on $\alpha$ if whenever $\alpha$ is invalidated by removing axioms from $\mathcal{O}^L_{\mathsf{temp}}$, then $\beta$ is also invalidated.

**Definition 4.** *Let $\mathcal{O} \models \alpha, \beta$. The axiom $\beta$ depends on $\alpha$ w.r.t. $\mathcal{O}$, written $\alpha \triangleright \beta$ iff for each $\mathcal{J}_\beta \in \mathsf{Just}(\beta, \mathcal{O})$ there is $\mathcal{J}_\alpha \in \mathsf{Just}(\alpha, \mathcal{O})$ such that $\mathcal{J}_\alpha \subseteq \mathcal{J}_\beta$. We say that $\alpha$ is a $\triangleright$-minimum (respectively $\triangleright$-maximum) if there is no $\beta$ s.t. $\alpha$ depends on $\beta$ (respectively $\beta$ depends on $\alpha$).*

The relation $\triangleright$ is consistent with our intuitions as shown next:

**Proposition 5.** *Let $\mathcal{O} \models \alpha, \beta$, $\mathcal{O}' \subset \mathcal{O}$ and $\alpha \triangleright \beta$. Then: 1) $\mathcal{O}' \not\models \alpha$ implies $\mathcal{O}' \not\models \beta$, and 2) $\mathcal{O}' \models \beta$ implies $\mathcal{O}' \models \alpha$.*

Figure 3(a) shows the GUI for selecting $\Im^-$. A similar interface is used to select $\Im^+$. The left-hand-side of the figure displays the dependency tree, which can be expanded and contracted in the usual way; on the right-hand side, users can select an entailment to remove and show its justifications. The justifications for the entailment highlighted in Figure 3(a) are shown in Figure 3(b). The GUI indicates which axioms in these justifications were selected in Step 2 from $\mathcal{O}^L$ and from $\mathcal{O}^R$, marking them with 'L' and 'R' respectively. Unmarked axioms occur in both ontologies.

### 4.4 Plan generation, selection and execution

Changing the set of entailments involves modifying the ontology itself. In general, there may be zero or more possible choices of sets of axioms to add and/or remove in order to satisfy a set of requirements. We call each of these choices a *repair plan* (or *plan*).

**Definition 6.** *Let $\mathcal{O}$, $\Im^+$, $\Im^-$, $\mathcal{O}^+$ and $\mathcal{O}^-$ be finite sets of axioms such that $\mathcal{O}^- \subseteq \mathcal{O}$, $\mathcal{O}^+ \cap \mathcal{O} = \emptyset$, $\mathcal{O} \models \Im^-$, $\mathcal{O} \cup \mathcal{O}^+ \models \Im^+$, and $\mathcal{O} \not\models \alpha$ for each $\alpha \in \Im^+$. A repair plan for $\mathcal{O}$ given $\mathcal{O}^+$, $\mathcal{O}^-$ $\Im^+$ and $\Im^-$ is a pair $\mathcal{P} = \langle \mathcal{P}^+, \mathcal{P}^- \rangle$ such that $\mathcal{P}^+ \subseteq \mathcal{O}^+$, $\mathcal{P}^- \subseteq \mathcal{O}^-$ and the following conditions hold:* 1) *$(\mathcal{O} \cup \mathcal{P}^+) \setminus \mathcal{P}^- \models \alpha$ for each $\alpha \in \Im^+$, and* 2) *$(\mathcal{O} \cup \mathcal{P}^+) \setminus \mathcal{P}^- \not\models \beta$ for each $\beta \in \Im^-$. $\mathcal{P}$ is minimal if there is no $\mathcal{P}_1$ s.t. $\mathcal{P}_1^+ \subset \mathcal{P}^+$ and $\mathcal{P}_1^- \subset \mathcal{P}^-$.*

Definition 6 extends the notion of a plan proposed in the ontology repair literature (e.g., see [4]). In particular, the goal of a plan in [4] is always to remove a set of axioms so that certain entailments do not hold anymore; hence, a plan is always guaranteed to exist. In contrast, a plan as in Definition 6 also involves adding axioms so that certain entailments hold; therefore, possibly conflicting sets of constraints need to be satisfied.

Step 8 involves computing all minimal plans (denoted $\mathbb{P}$). In our case, the ontology $\mathcal{O}$ to be repaired is $\mathcal{O}^L_{\mathsf{temp}}$ from Step 3. The intended and unintended entailments ($\Im^+$ and $\Im^-$) are those selected in Steps 6 and 7. We assume that a plan can add any subset of the axioms in $\Lambda_s$ not originally selected in Step 2 (i.e. $\mathcal{O}^+ = \Lambda_s \setminus \mathcal{S}$). A plan can remove any subset of the selected axioms (i.e., $\mathcal{O}^- = \mathcal{S}$). Hence, we assume that plans should not remove axioms outside $\Lambda_s$, which are common to both versions of the ontology.

In Step 9 users can select from $\mathbb{P}$ a plan to be applied. If no plan matches their intentions, the conflict resolution process ends as it started; that is, by returning the old version of $\mathcal{O}^L$ in a conflicting state (Step 9). In contrast, if a plan $\mathcal{P}$ is selected, then $\mathcal{P}$ is applied by returning the ontology $(\mathcal{O}^L_{\mathsf{temp}} \cup \mathcal{P}^+) \setminus \mathcal{P}^-$ in a non-conflicting state (Steps 10–11), which is then ready to be committed.

ContentCVS implements a plan computation algorithm (see [8]) that reuses the justifications already computed when obtaining the orderings from Definition 4. The algorithm prunes the set of possible plans using the following principles: *(i)* in order for an entailment $\alpha \in \Im^+$ to hold after the execution of a plan $\langle \mathcal{P}^+, \mathcal{P}^- \rangle$, $(\mathcal{O} \cup \mathcal{P}^+) \setminus \mathcal{P}^-$ must contain at least one justification for $\alpha$ in $\mathcal{O} \cup \mathcal{O}^+$; *(ii)* in order for an entailment $\beta \in \Im^-$ not to hold after the execution of a plan $\langle \mathcal{P}^+, \mathcal{P}^- \rangle$, it is sufficient to show that no justification for $\beta$ in $\mathcal{O} \cup \mathcal{O}^+$ is contained in $(\mathcal{O} \cup \mathcal{P}^+) \setminus \mathcal{P}^-$.

ContentCVS also provides functionality for selecting the most suitable minimal plan (Step 10). In particular, it identifies the axioms in the structural difference of $\mathcal{O}^L$

**Input:** $\mathcal{O}, \mathcal{O}'$: ontologies; approximation $\mathsf{diff}_{\approx}$
Compute $\Lambda_s$ and store its size (**I**) and computation time (**II**)
**repeat**
Randomly select $\mathcal{S} \subseteq \Lambda_s$, and compute $\mathcal{O}_{\mathsf{aux}} := \mathcal{O} \cup \mathcal{S}$
Compute $\mathsf{diff}_{\approx}(\mathcal{O}_{\mathsf{aux}}, \mathcal{O}) \cup \mathsf{diff}_{\approx}(\mathcal{O}_{\mathsf{aux}}, \mathcal{O}')$ and store its size (**III**)
Compute $\mathsf{diff}_{\approx}(\mathcal{O}, \mathcal{O}_{\mathsf{aux}})$ and store its size (**IV**)
Compute all justifications for entailments in $\mathsf{diff}_{\approx}$ and store avg. time per justification (**V**)
Compute $\triangleright$, and store the number of $\triangleright$-minimums (**VI**)
Randomly select $\Im^-$ from minimums of $\triangleright$ and $\Im^+$ from maximums of $\triangleright$
Compute $\mathbb{P}$ (minimal plans) and store number of plans (**VII**) and computation time (**VIII**)
**until** 200 iterations have been performed

**Table 2.** Synthetic Experiments

and $\mathcal{O}^R$ that are shared by all minimal plans, and presents the remaining axioms in a separate frame, allowing users to select which ones among them a plan must either add or delete. The tool then filters the minimal plans according to the user's selections.

## 5 Evaluation

We have evaluated the *performance* of the implemented algorithms by conducting a number of synthetic experiments based on the evolution of a realistic ontology. We have evaluated the *usability* of ContentCVS, and the *adequacy of our approach* for ontology development by conducting a pilot user study.

### 5.1 Synthetic experiments

We have simulated the evolution of a medical ontology by using a sequence of 11 versions of an ontology developed at the University of Manchester[6]. Their size varies from 71 classes, 13 roles and 195 axioms in the first version to 207 concepts, 38 roles and 620 axioms in the last version. Their DL expressivity is $\mathcal{SHIQ}(D)$. The experiments were performed on a laptop computer with a 1.82 GHz processor and 3Gb of RAM.

For each pair $\mathcal{O}_i, \mathcal{O}_{i+1}, i \in \{1, \ldots, 10\}$ of consecutive versions, and both the smallest and largest approximations of the deductive difference in ContentCVS, we have performed the synthetic experiment in Table 2, where the Roman numbers refer to measurements stored during the experiment, and presented in Table 3. These experiments follow our approach for conflict resolution in Table 1, with the assumption that $\mathcal{O}_i$ is the local ontology, $\mathcal{O}_{i+1}$ is the repository ontology, and the steps in Table 1 requiring manual intervention are performed randomly. Table 3 summarises our results[7].

Several conclusions can be drawn. First the main computational bottleneck is the computation of all the justifications for relevant entailments. Once the justifications have been computed, the computation of the plans is relatively fast. Second, the amount

[6] Thanks to Alan Rector for providing this test sequence.

[7] `http://krono.act.uji.es/people/Ernesto/contentcvs/synthetic-study`

| | | | Smallest diff$_\approx$ approximation | | | | | | Largest diff$_\approx$ approximation | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **I** | **II** | **III** | **IV** | **V** | **VI** | **VII** | **VIII** | **III** | **IV** | **V** | **VI** | **VII** | **VIII** |
| $\mathcal{O}$ **&** $\mathcal{O}'$ | | | avg | avg | avg | avg | avg/max | avg | avg | avg | avg | avg | avg/max | avg |
| $\mathcal{O}_1$&$\mathcal{O}_2$ | 50 | 0.03 | 15 | 6 | 0.1 | 15 | 1 / 1 | 1.5 | 111 | 17 | 2.0 | 33 | 495 / 5508 | 10.3 |
| $\mathcal{O}_3$&$\mathcal{O}_4$ | 82 | 0.02 | 13 | 4 | 0.26 | 13 | 3 / 18 | 1.7 | 128 | 90 | 0.9 | 30 | 46 / 896 | 3.6 |
| $\mathcal{O}_5$&$\mathcal{O}_6$ | 93 | 0.02 | 31 | 14 | 0.1 | 29 | 3 / 32 | 1.2 | 267 | 48 | 5.9 | 49 | 2.7 / 6 | 30 |
| $\mathcal{O}_8$&$\mathcal{O}_9$ | 110 | 0.03 | 19 | 15 | 0.02 | 18 | 1 / 4 | 0.07 | 216 | 78 | 1.2 | 47 | 488 / 3888 | 4 |
| $\mathcal{O}_9$&$\mathcal{O}_{10}$ | 79 | 0.02 | 15 | 6 | 0.06 | 14 | 1 / 2 | 0.3 | 251 | 14 | 3.7 | 46 | 101 / 720 | 21.5 |
| $\mathcal{O}_{10}$&$\mathcal{O}_{11}$ | 117 | 0.01 | 24 | 8 | 1.5 | 24 | 7 / 50 | 15.6 | 208 | 154 | 5.3 | 31 | 35 / 225 | 22.7 |

**Table 3.** Summary of Results. Roman numbers refer to Table 2. Time measures given in seconds

of information presented to the user largely depends on the selected approximation for the deductive difference. For the smallest approximation, the average number of axioms in the relevant differences (see **III** and **IV**) is in the range $4$–$31$, and the average number of minimal plans (see **VII**) is in the range $1$–$50$. In contrast, in the case of the largest approximation, these average numbers are in the ranges $14$–$267$, and $6$–$5508$ respectively. The amount of information the user would need to consider is thus much larger. Table 3 also shows that the use of the dependency relation $\triangleright$ can lead to a significant reduction in the amount of presented information (**VI**).

### 5.2 User study

We have conducted a pilot user study to evaluate the usability of ContentCVS, as well as to show the adequacy of our approach in practice. The details of the study, including the questionnaire and the test ontologies, are available online. [8]

The study consists of three parts. *Part 1* simulates a conventional debugging scenario where a single developer changes his/her ontology $\mathcal{O}_0$ and, as a result, creates a new version $\mathcal{O}_1$ of the ontology in which errors have been introduced. *Part 2* simulates the scenario where a single developer working with $\mathcal{O}^L$ performs a CVS-update and needs to reconcile $\mathcal{O}^L$ with the version $\mathcal{O}^R$ in the repository using our methodology in Table 1 from Section 4. Finally, *Part 3* simulates the collaborative development of an ontology. Each participant extended an initial ontology by performing a-priori specified changes, and tried to execute a CVS-commit either after completing the changes, or when explicitly indicated. If the commit failed, the participants had to update and reconcile the changes using their ContentCVS client. In the end, users discussed the final reconciled ontology among themselves and with the coordinator of the study.

We had eight participants in Parts 1 and 2, and conducted three collaborative tests in Part 3. They were academic researchers, most of them working in fields other than semantic Web. Most users evaluated their experience on DLs and OWL as either 'intermediate' or 'low', but had used both Protégé and a CVS file system before.

Our results show that most users considered very useful the computation of structural differences between ontology versions, but found it difficult to detect errors by examining only the structural difference.

Users liked the detection of errors using approximations of the deductive difference; when using ContentCVS, everyone could identify both new unintended entailments and lost intended entailments. Most participants were satisfied with the smallest

[8] `http://krono.act.uji.es/people/Ernesto/contentcvs/user-study`

approximation implemented in ContentCVS, and complained about excessive amount of displayed information when using the largest implemented approximation. Interestingly, by using our largest approximation they could detect errors other than unsatisfiable concepts and atomic subsumptions. The computation of the dependency relation ($\triangleright$) was evaluated very positively, but users complained about the response time.

Most users were satisfied with the functionality for computing repair plans as well as with the ontology obtained after the execution of the selected plan. Users also described the CVS functionality in ContentCVS as either 'very useful' or 'useful' and evaluated the tool's workflow very positively. In particular, most of them evaluated the GUI as 'good' and the ontology development workflow as either 'very good' or 'good'.

The main points of criticism were, on the one hand, the excessive amount of information displayed when using 'large' approximations of the deductive difference and, on the other hand, slow response of the tool when computing all justifications of certain entailments and/or computing large approximations of the deductive difference. We consider addressing these deficiencies as a part of our future work.

## 6 Related and future work

There have been several recent proposals for facilitating collaboration in ontology engineering tools. Collaborative Protégé [14, 15] allows developers to hold discussions, chat, and annotate changes. The authors of [16] present an ontology change management system which works similarly to a CVS. The change history is stored in a server and the system can identify differences in the change sets from different clients. The functionality described in [14, 15, 16] and our techniques naturally complement each other. For example, discussion threads and annotations could be used in ContentCVS to assist users in selecting intended and unintended consequences (i.e. in Steps 9 and 10 of Table 1), and for recording the rationale behind their selections.

The authors of [17] propose a 'locking' mechanism that allows a user, for example, to establish a lock over a concept, meaning that other users are not allowed to make changes that 'affect' that concept until the lock has been released. Although errors can still occur, the idea is that these locks would mitigate them. The precise guarantees provided by locks are, however, not clear. Conflicts are likely to arise, and the approach in [17] does not provide means for detecting and resolving them if they do.

The OWLDiff tool [9] provides a GUI for computing the deductive differences between pairs of OWL 2 EL ontologies. The tool, however, does not provide means for explaining these differences, resolving conflicts or constructing the reconciled ontology.

In the future, we plan to improve the system's performance and in particular the computation of justifications. We also aim at integrating in our tool some of the functionality provided by Collaborative Protégé for holding discussions and annotate changes. Finally, given the encouraging output of the evaluation, we are planning to apply our results in a real-world scenario in the context of the HeC project.

[9] OWLDiff: `http://sourceforge.net/projects/owldiff`

## References


[1] Hartung, M., Kirsten, T., Rahm, E.: Analyzing the evolution of life science ontologies and mappings. In: Proc. of DILS. (2008) 11–27

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

[3] Kontchakov, R., Wolter, F., Zakharyaschev, M.: Can you tell the difference between DL-Lite ontologies? In: Proc. of KR, AAAI Press (2008) 285–295

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

[5] Schlobach, S., Huang, Z., Cornet, R., van Harmelen, F.: Debugging incoherent terminologies. J. Autom. Reasoning **39**(3) (2007) 317–349

[6] Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. J. Web Semantics **3**(4) (2005) 268–293

[7] Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Proc. of ISWC. (2007)

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

[9] Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The next step for OWL. J. Web Semantics **6**(4) (2008) 309–322

[10] Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language: Profiles. W3C Working Draft (2009) http://www.w3.org/TR/owl2-profiles/.

[11] Motik, B., Patel-Schneider, P., B.Parsia: OWL 2 web ontology language structural specification and functional-style syntax. W3C Working Draft (2009) http://www.w3.org/TR/owl2-syntax/.

[12] Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: extracting modules from ontologies. In: Proc. of WWW. (2007) 717–726

[13] Suntisrivaraporn, B., Qi, G., Ji, Q., Haase, P.: A modularization-based approach to finding all justifications for OWL DL entailments. In: Proc. of ASWC. 1–15

[14] Noy, N.F., Tudorache, T., de Coronado, S., Musen, M.A.: Developing biomedical ontologies collaboratively. In: Proc. of AMIA 2008. (2008)

[15] Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A framework for ontology evolution in collaborative environments. In: Proc. of ISWC. (2006) 544–558

[16] Redmond, T., Smith, M., Drummond, N., Tudorache., T.: Managing change: An ontology version control system. In: Proc. of OWLEd. (2008)

[17] Seidenberg, J., Rector, A.L.: A methodology for asynchronous multi-user editing of semantic web ontologies. In: Proc. of K-CAP. (2007) 127–134