Owl2vcs: Tools for Distributed Ontology Development

Ivan Zaikin, Anatoly Tuzovsky

Tomsk Polytechnic University, Tomsk, Russia
`{i,tuzovskyaf}@tpu.ru`

Abstract. The collaborative development of web ontologies is an important topic being actively researched. In this paper, we present a set of tools that facilitate collaborative development of web ontologies using distributed version control systems. The main purpose of these tools is to replace inefficient built-in diff and three-way merge tools, which rely on text representation rather than on ontology structure. Being based on popular distributed version control systems, owl2vcs can be easily integrated into popular issue trackers.

Keywords: OWL 2, ontology, structural difference, three-way merge, change, change set.

1 Introduction

Ontology is a formal description of a specific domain in a standardized language readable by both humans and computers. Like software, ontologies need to be maintained after development. Managing changes between ontologies is an essential part of ontology engineering. There are many reasons for ontologies to be changed. Firstly, the domain may change. Secondly, the understanding of the domain by ontology engineers may change. Thirdly, there may be errors to fix and new ways of representing the domain in an ontology language. Development and maintenance of large ontologies are performed by groups of specialists. Nowadays collaborative development of software is inconceivable without using revision control and issue tracking systems. Using the same approach for ontology development is very promising but has some pitfalls. All tools designed for software code development are text-oriented. For example, diff and three-way merge tools are useless for ontologies stored in one of XML formats.

The OWL 2 language has two semantics: 1) RDF-based semantics where resources and relations between them are the main concepts, and complex constructs are represented using so called “blank nodes”, and 2) direct semantics where entities and axioms are the main concepts. There are works related to comparison of RDF graphs but they either do not mention the blank nodes problem, or the algorithms are too complex and non-deterministic [1]. It is shown in [2] that in the general case it is impossible to compare unambiguously two RDF triples containing blank nodes, though there are computationally complex algorithms of checking whether two RDF graphs are isomorphic. In this paper, we present algorithms and structural[1] diff and three-way merge tools for OWL 2 ontologies, which use OWL 2 direct semantics. Applying direct semantics eliminates problems with blank nodes and allows comparing ontologies axiom by axiom. They also take into account some attributes, which are not included into the logical constituent but still important for ontology development.

2 Related Work

The development of owl2vcs was inspired by SVoNt [3] – an SVN-based ontology versioning system. However, it is not available for download and looks like to be abandoned.

There are also some tools for comparing ontologies using OWL direct semantics:

1. OWLPatch [4] is a simple command line tool, which compares two ontologies and saves the result for future patching of the first ontology.
2. Ecco [5] is a system, which compares two ontologies and analyzes the changes detecting whether removals and additions were logically effectual.
3. OWLDiff [6] is a tool, which compares ontologies axiom by axiom, finds the effectual changes and analyzes them to find the affected entities.
4. OWL Diff Tool [7] is plug-in for Protégé 4.2, which calculates the structural differences between ontologies and focuses on alignment of renamed entities.

The tools mentioned above only compare the logical constituent of ontologies (i.e. axioms) and do not take into account import changes, prefix changes, and ontology format changes. However, such changes are important for ontology developers and it is useful to keep record of them. During the research, we found no tool capable of three-way merge of OWL ontologies. That is why we have developed our own one.

3 Algorithms

Let us denote a set of all OWL 2 ontologies as $\mathbf{O}^U$, and introduce the notation for ontology components:

- $\mathbf{E}^U$ – set of all possible OWL 2 entities (classes, data types, individuals and properties);
- $\mathbf{A}^U$ – set of all possible OWL 2 axioms (logical statements about entities);
- $\mathbf{N}^U$ – set of all possible ontology annotations – pairs of the form (n_p, n_v), where n_p is an annotation property and n_v is its value;
- $\mathbf{I}^U$ – set of all possible imports (links to other ontologies);
- $\mathbf{P}^U \subset \mathbf{S} \times \mathbf{S}$ – functional binary relation defined on the set of strings $\mathbf{S}$ representing the set of all possible prefixes – pairs (p_n, p_i), where p_i – prefix value – is a part of

[1] The notion of structural difference is borrowed from [5] where it is defined through the notion of structural equivalence used in [8].

IRI, and p_n – prefix name – is a short string used to shorten names of entities and can be used instead of p_n;

- **D** – set of all possible pairs (r_o, r_v) called ontology identifiers, where r_o is an ontology IRI, and r_v is a version IRI;
- **F** = { «RDF/XML», «Turtle», «OWL/XML», «OWL Functional Syntax», «OWL Manchester Syntax» } – set of possible ontology formats.

An ontology is a tuple $\langle \mathbf{E}, \mathbf{A}, \mathbf{N}, \mathbf{I}, \mathbf{P}, d, f \rangle$, where

- $\mathbf{E} \subset \mathbf{E}^{U}$ – finite unordered set of entities;
- $\mathbf{A} \subset \mathbf{A}^{U}$ – finite unordered set of axioms;
- $\mathbf{N} \subset \mathbf{N}^{U}$ – finite unordered set of ontology annotations;
- $\mathbf{I} \subset \mathbf{I}^{U}$ – finite unordered set of imports;
- $\mathbf{P} \subset \mathbf{P}^{U}$ – finite unordered set of prefixes;
- $d \in \mathbf{D}$ – ontology identifies;
- $f \in \mathbf{F}$ – ontology format.

An ontology can only contain those entities, which are referred by at least one axiom. Consequently, removing an entity from an ontology implies removing all axioms referred to that entity; adding an entity implies adding at least one axiom referring to that entity; changing an entity implies adding an axiom which refers to that entity and is not the first one, or removing an axiom which refers to that entity and is not the last one.

Let us denote changes in each component:

- $\mathbf{C}^{U} = \mathbf{C}_{A}^{U} \cup \mathbf{C}_{N}^{U} \cup \mathbf{C}_{I}^{U} \cup \mathbf{C}_{P}^{U} \cup \mathbf{C}_{d}^{U} \cup \mathbf{C}_{f}^{U}$ – set of all possible changes;
- $\mathbf{C}_{A}^{U} = \mathbf{C}_{A}^{U+} \cup \mathbf{C}_{A}^{U-}$ – set of all possible axiom changes;
- $\mathbf{C}_{A}^{U+} = \{ \text{AddAxiom}(a) \mid a \in \mathbf{A}^{U} \}$ – set of all possible axiom additions;
- $\mathbf{C}_{A}^{U-} = \{ \text{RemoveAxiom}(a) \mid a \in \mathbf{A}^{U} \}$ – set of all possible axiom removals;
- $\mathbf{C}_{N}^{U} = \mathbf{C}_{N}^{U+} \cup \mathbf{C}_{N}^{U-}$ – set of all possible ontology annotation changes;
- $\mathbf{C}_{N}^{U+} = \{ \text{AddOntologyAnnotation}(n) \mid n \in \mathbf{N}^{U} \}$ – set of all possible ontology annotation additions;
- $\mathbf{C}_{N}^{U-} = \{ \text{RemoveOntologyAnnotation}(n) \mid n \in \mathbf{N}^{U} \}$ – set of all possible ontology annotation removals;
- $\mathbf{C}_{I}^{U} = \mathbf{C}_{I}^{U+} \cup \mathbf{C}_{I}^{U-}$ – set of all possible import changes;
- $\mathbf{C}_{I}^{U+} = \{ \text{AddImport}(i) \mid i \in \mathbf{I}^{U} \}$ – set of all possible import additions;
- $\mathbf{C}_{I}^{U-} = \{ \text{RemoveImport}(i) \mid i \in \mathbf{I}^{U} \}$ – set of all possible import removals;
- $\mathbf{C}_{P}^{U} = \mathbf{C}_{P}^{U+} \cup \mathbf{C}_{P}^{U-} \cup \mathbf{C}_{P}^{U*} \cup \mathbf{C}_{P}^{U\#}$ – set of all possible prefix changes;
- $\mathbf{C}_{P}^{U+} = \{ \text{AddPrefix}(p_n, p_i) \mid (p_n, p_i) \in \mathbf{P}^{U} \}$ – set of all possible prefix additions;
- $\mathbf{C}_{P}^{U-} = \{ \text{RemovePrefix}(p_n, p_i) \mid (p_n, p_i) \in \mathbf{P}^{U} \}$ – set of all possible prefix removals;
- $\mathbf{C}_{P}^{U*} = \{ \text{ModifyPrefix}(p_n, p_i, p_i') \mid (p_n, p_i) , (p_n, p_i') \in \mathbf{P}^{U} \wedge p_i \neq p_i' \}$ – set of all possible prefix modifications;
- $\mathbf{C}_{P}^{U\#} = \{ \text{RenamePrefix}(p_n, p_i, p_n') \mid (p_n, p_i), (p_n', p_i) \in \mathbf{P}^{U} \wedge p_n \neq p_n' \}$ – set of all possible prefix renames;
- $\mathbf{C}_{d}^{U} = \{ \text{SetOntologyID}(d, d') \mid d, d' \in \mathbf{D} \wedge d \neq d' \}$ – set of all possible ontology identifier changes;

- $\mathbf{C}_f^{\mathrm{U}}$ = { SetOntologyFormat(f, f') | $f, f' \in \mathbf{F} \wedge f \neq f'$ }– set of all possible ontology format changes.

3.1 Ontology Diff

Let $v_1 = \langle \mathbf{E}_1, \mathbf{A}_1, \mathbf{N}_1, \mathbf{I}_1, \mathbf{P}_1, d_1, f_1 \rangle \in \mathbf{O}^{\mathrm{U}}$ be the first version of an ontology and $v_2 = \langle \mathbf{E}_2, \mathbf{A}_2, \mathbf{N}_2, \mathbf{I}_2, \mathbf{P}_2, d_2, f_2 \rangle \in \mathbf{O}^{\mathrm{U}}$ – the second one; $\delta : \mathbf{O}^{\mathrm{U}} \times \mathbf{O}^{\mathrm{U}} \rightarrow \mathcal{P}(\mathbf{C}^{\mathrm{U}})$ is a function that returns the change set $\mathbf{C}$ between two versions; $\varepsilon : \mathbf{O}^{\mathrm{U}} \times (\mathbf{C}^{\mathrm{U}}) \rightarrow \mathbf{O}^{\mathrm{U}}$ is a function which applies the change set $\mathbf{C}$ to the ontology. The problem of comparing two versions v_1 and v_2 is to find such a change set $\mathbf{C} = \delta(v_1, v_2)$ that $\varepsilon(v_1, \mathbf{C}) = v_2$.
Let's denote the functions which compare constituents of ontologies:

- $\delta_{\mathrm{A}} : \mathcal{P}(\mathbf{A}^{\mathrm{U}}) \times \mathcal{P}(\mathbf{A}^{\mathrm{U}}) \rightarrow \mathcal{P}(\mathbf{C}_{\mathrm{A}}^{\mathrm{U}})$ – the axiom comparison function;
- $\delta_{\mathrm{N}} : \mathcal{P}(\mathbf{N}^{\mathrm{U}}) \times \mathcal{P}(\mathbf{N}^{\mathrm{U}}) \rightarrow \mathcal{P}(\mathbf{C}_{\mathrm{N}}^{\mathrm{U}})$ – the ontology annotation comparison function;
- $\delta_{\mathrm{I}} : \mathcal{P}(\mathbf{I}^{\mathrm{U}}) \times \mathcal{P}(\mathbf{I}^{\mathrm{U}}) \rightarrow \mathcal{P}(\mathbf{C}_{\mathrm{I}}^{\mathrm{U}})$ – the import comparison function;
- $\delta_{\mathrm{P}} : \mathcal{P}(\mathbf{P}^{\mathrm{U}}) \times \mathcal{P}(\mathbf{P}^{\mathrm{U}}) \rightarrow \mathcal{P}(\mathbf{C}_{\mathrm{P}}^{\mathrm{U}})$ – the prefix comparison function;
- $\delta_d : \mathbf{D} \times \mathbf{D} \rightarrow \mathcal{P}_{\leq 1}(\mathbf{C}_d^{\mathrm{U}})$ – the ontology identifier comparison function;
- $\delta_f : \mathbf{F} \times \mathbf{F} \rightarrow \mathcal{P}_{\leq 1}(\mathbf{C}_f^{\mathrm{U}})$ – the ontology format comparison function.

The change set between v_1 and v_2 is a union of changes in separate constituents:

$$\mathbf{C} = \delta(v_1, v_2) =$$
$$= \delta_{\mathrm{A}}(\mathbf{A}_1, \mathbf{A}_2) \cup \delta_{\mathrm{N}}(\mathbf{N}_1, \mathbf{N}_2) \cup \delta_{\mathrm{I}}(\mathbf{I}_1, \mathbf{I}_2) \cup \delta_{\mathrm{P}}(\mathbf{P}_1, \mathbf{P}_2) \cup \delta_d(d_1, d_2) \cup \delta_f(f_1, f_2) =$$
$$= \mathbf{C}_{\mathrm{A}} \cup \mathbf{C}_{\mathrm{N}} \cup \mathbf{C}_{\mathrm{I}} \cup \mathbf{C}_{\mathrm{P}} \cup \mathbf{C}_d \cup \mathbf{C}_f$$

Axiom changes can be found by comparing two sets $\mathbf{A}_1$ and $\mathbf{A}_2$:

$$\mathbf{C}_{\mathrm{A}} = \delta_{\mathrm{A}}(\mathbf{A}_1, \mathbf{A}_2) =$$
$$= \{ \text{AddAxiom}(a) \mid a \in \mathbf{A}_2 \setminus \mathbf{A}_1 \} \cup \{ \text{RemoveAxiom}(a) \mid a \in \mathbf{A}_1 \setminus \mathbf{A}_2 \}.$$

That is, added axioms are axioms which present in $\mathbf{A}_2$ but absent in $\mathbf{A}_1$, and removed axioms are those which present in $\mathbf{A}_1$ but absent in $\mathbf{A}_2$.

Ontology annotation changes can be found in a similar way:

$$\mathbf{C}_{\mathrm{N}} = \delta_{\mathrm{N}}(\mathbf{N}_1, \mathbf{N}_2) =$$
$$= \{ \text{AddOntologyAnnotation}(n) \mid n \in \mathbf{N}_2 \setminus \mathbf{N}_1 \} \cup$$
$$\cup \{ \text{RemoveOntologyAnnotation}(n) \mid n \in \mathbf{N}_1 \setminus \mathbf{N}_2 \}.$$

Import changes are obtained similarly:

$$\mathbf{C}_{\mathrm{I}} = \delta_{\mathrm{I}}(\mathbf{I}_1, \mathbf{I}_2) = \{ \text{AddImport}(i) \mid i \in \mathbf{I}_2 \setminus \mathbf{I}_1 \} \cup \{ \text{RemoveImport}(i) \mid i \in \mathbf{I}_1 \setminus \mathbf{I}_2 \}.$$

Prefix changes $\mathbf{C}_{\mathrm{P}}$ fall into four categories: additions, removals, modifications and renames:

$$\mathbf{C}_{\mathrm{P}} = \delta_{\mathrm{P}}(\mathbf{P}_1, \mathbf{P}_2) =$$
$$= \{ \text{AddPrefix}(p_{n2}, p_{i2}) \mid (p_{n2}, p_{i2}) \in \mathbf{P}_2 \wedge \nexists (p_{n1}, p_{i1}) \in \mathbf{P}_1\text{: } (p_{n1}=p_{n2} \vee p_{i1}=p_{i2}) \} \cup$$
$$\cup \{ \text{RemovePrefix}(p_{n1}, p_{i1}) \mid (p_{n1}, p_{i1}) \in \mathbf{P}_1 \wedge \nexists (p_{n2}, p_{i2}) \in \mathbf{P}_2\text{: } (p_{n1}=p_{n2} \vee p_{i1}=p_{i2}) \} \cup$$
$$\cup \{ \text{ModifyPrefix}(p_{n1}, p_{i1}, p_{i2}) \mid (p_{n1}, p_{i1}) \in \mathbf{P}_1 \wedge \exists (p_{n2}, p_{i2}) \in \mathbf{P}_2\text{: } (p_{n1}=p_{n2} \wedge p_{i1} \neq p_{i2}) \} \cup$$
$$\cup \{ \text{RenamePrefix}(p_{n1}, p_{i1}, p_{n2}) \mid$$
$$(p_{n1}, p_{i1}) \in \mathbf{P}_1 \wedge \exists (p_{n2}, p_{i2}) \in \mathbf{P}_2\text{: } ((p_{n2}, p_{i2}) \notin \mathbf{P}_1) \wedge p_{n1} \neq p_{n2} \wedge p_{i1}=p_{i2} \}.$$

Thus prefixes in $\mathbf{P}_2$ but not in $\mathbf{P}_1$ are considered added; prefixes in $\mathbf{P}_1$ but not in $\mathbf{P}_2$ are considered removed; prefixes with different values but the same name are considered modified; and prefixes with different names but the same values are considered renamed.

The δ_d function returns the set of at most one element which describes the change of the ontology identifier:

$$\mathbf{C}_d = \delta_d(d_1, d_2) = \{ \text{SetOntologyID}(d_1, d_2) \mid d_1 \neq d_2 \}.$$

The δ_f function returns the set of at most one element which describes the change of the ontology format:

$$\mathbf{C}_f = \delta_f(f_1, f_2) = \{ \text{SetOntologyFormat}(f_1, f_2) \mid f_1 \neq f_2 \}.$$

3.2 Analysis of Changes

Let the first version of an ontology v_1 contain a set of entities $\mathbf{E}_1$, and the second version v_2 contain another set of entities $\mathbf{E}_2$. The problem of change analysis is to do the following:

5. Find the set of entities $\mathbf{E}' \subset \mathbf{E}_1 \cup \mathbf{E}_2$ affected by changes;
6. Split the set $\mathbf{E}'$ into three subsets: $\mathbf{E}^+$ – new entities; $\mathbf{E}^-$ – removed entities; $\mathbf{E}^*$ – modified entities (which present in both versions);
7. For each new, removed, or modified entity, find a set of corresponding changes.

Let's define the signature $\sigma(a)$ of axiom a as a finite unordered set of entities which are referred by axiom a. The signature of a set of axioms can be found as a union of separate axiom signatures:

$$\sigma(\{a_1, a_2, \ldots, a_n\}) = \sigma(a_1) \cup \sigma(a_2) \cup \ldots \cup \sigma(a_n).$$

The signature of a change is equal to the signature of the corresponding axiom:

$$\sigma(\text{AddAxiom}(a)) = \sigma(a),\ \sigma(\text{RemoveAxiom}(a)) = \sigma(a).$$

The signature of a change set can be found as a union of single change signatures:

$$\sigma(\{c_1, c_2, \ldots, c_n\}) = \sigma(c_1) \cup \sigma(c_2) \cup \ldots \cup \sigma(c_n).$$

The set of entities affected by changes $\mathbf{C}_\text{A}$ can be found as a signature of $\mathbf{C}_\text{A}$: $\mathbf{E}' = \sigma(\mathbf{C}_\text{A})$. It is shown in Figure 1 as a dashed rectangle. It follows from the figure that removed entities $\mathbf{E}^- = \mathbf{E}' \setminus \mathbf{E}_2 = \mathbf{E}_1 \setminus \mathbf{E}_2$, added entities $\mathbf{E}^+ = \mathbf{E}' \setminus \mathbf{E}_1 = \mathbf{E}_2 \setminus \mathbf{E}_1$, and modified entities $\mathbf{E}^* = \mathbf{E}' \cap \mathbf{E}_1 \cap \mathbf{E}_2$. The figure also shows unaffected entities $\mathbf{E}^{||} = \mathbf{E}_1 \cap \mathbf{E}_2 \setminus \mathbf{E}'$.

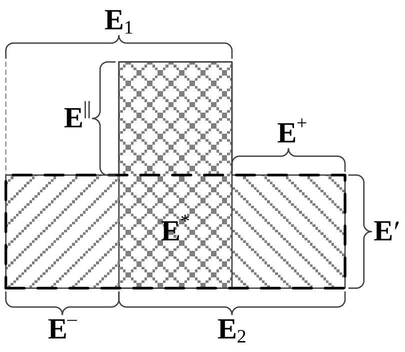


Fig. 1. Relationship between $\mathbf{E}_1$, $\mathbf{E}_2$, $\mathbf{E}^{+}$, $\mathbf{E}^{-}$, $\mathbf{E}^{*}$, $\mathbf{E}^{\parallel}$ and $\mathbf{E}'$

For each modified, removed or added entity e we can find all related changes as follows:

$$\sigma^{-1}(e) = \{ c \mid c \in \mathbf{C}_\mathrm{A} \wedge e \in \sigma(c) \}.$$

3.3 Three-way Merge of Ontologies

Let

- $v_0 = \langle \mathbf{E}_0, \mathbf{A}_0, \mathbf{N}_0, \mathbf{I}_0, \mathbf{P}_0, d_0, f_0 \rangle \in \mathbf{O}^\mathrm{U}$ be the initial version of an ontology;
- $v_1 = \langle \mathbf{E}_1, \mathbf{A}_1, \mathbf{N}_1, \mathbf{I}_1, \mathbf{P}_1, d_1, f_1 \rangle \in \mathbf{O}^\mathrm{U}$ – the version changed by the first user;
- $v_2 = \langle \mathbf{E}_2, \mathbf{A}_2, \mathbf{N}_2, \mathbf{I}_2, \mathbf{P}_2, d_2, f_2 \rangle \in \mathbf{O}^\mathrm{U}$ – the version changed by the second user;
- $\mathbf{C}_1 = \delta(v_0, v_1) = \mathbf{C}_{1\mathrm{A}} \cup \mathbf{C}_{1\mathrm{N}} \cup \mathbf{C}_{1\mathrm{I}} \cup \mathbf{C}_{1\mathrm{P}} \cup \mathbf{C}_{1d} \cup \mathbf{C}_{1f}$ – finite unordered set of changes made to v_0 by the first user;
- $\mathbf{C}_2 = \delta(v_0, v_2) = \mathbf{C}_{2\mathrm{A}} \cup \mathbf{C}_{2\mathrm{N}} \cup \mathbf{C}_{2\mathrm{I}} \cup \mathbf{C}_{2\mathrm{P}} \cup \mathbf{C}_{2d} \cup \mathbf{C}_{2f}$ – finite unordered set of changes made to v_0 by the second user.

The changes made by both users $\mathbf{C}_1 \cup \mathbf{C}_2$ can be split into following subsets:

- Matching changes $\mathbf{C}_m = \mathbf{C}_1 \cap \mathbf{C}_2$;
- Conflicting changes by the first user: $\mathbf{C}'_1$;
- Non-conflicting changes by the first user: $\mathbf{C}^{\parallel}_1 = (\mathbf{C}_1 \setminus \mathbf{C}_m) \setminus \mathbf{C}'_1$;
- Conflicting changes by the second user: $\mathbf{C}'_2$;
- Non-conflicting changes by the second user: $\mathbf{C}^{\parallel}_2 = (\mathbf{C}_1 \setminus \mathbf{C}_m) \setminus \mathbf{C}'_2$.

The problem of three-way merging of ontologies v_0, v_1 and v_2 is to get the v_3 ontology based on v_0 and containing changes $\mathbf{C}_m$, $\mathbf{C}^{\parallel}_1$, $\mathbf{C}^{\parallel}_2$ and some changes from $\mathbf{C}'_1 \cup \mathbf{C}'_2$ which don't lead to a conflict.

In this paper, we assume that import changes and ontology annotation changes cannot lead to a conflict. The conflicting changes therefore include conflicting changes of axioms, prefixes, ontology identifiers and ontology formats:

$$\mathbf{C}'_1 = \mathbf{C}'_{1\mathrm{A}} \cup \mathbf{C}'_{1\mathrm{P}} \cup \mathbf{C}'_{1d} \cup \mathbf{C}'_{1f};$$

$$\mathbf{C}'_2 = \mathbf{C}'_{2\mathrm{A}} \cup \mathbf{C}'_{2\mathrm{P}} \cup \mathbf{C}'_{2d} \cup \mathbf{C}'_{2f}.$$

The ontology format change is conflicting if another user has also changed the ontology format:

$$\mathbf{C}'_{1f} = \{\, c \mid c \in \mathbf{C}_{1f} \wedge c \notin \mathbf{C}_{2f} \,\};$$
$$\mathbf{C}'_{2f} = \{\, c \mid c \in \mathbf{C}_{2f} \wedge c \notin \mathbf{C}_{1f} \,\}.$$

Similarly, the ontology identifier change is conflicting if another user also changed the ontology identifier, and the change was not the same:

$$\mathbf{C}'_{1d} = \{\, c \mid c \in \mathbf{C}_{1d} \wedge c \notin \mathbf{C}_{2d} \,\};$$
$$\mathbf{C}'_{2d} = \{\, c \mid c \in \mathbf{C}_{2d} \wedge c \notin \mathbf{C}_{1d} \,\}.$$

Conflicting changes of axioms are non-matching changes made by different users and affecting same entities:

$$\mathbf{C}'_{1\mathrm{A}} = \{\, c \in \mathbf{C}_1 \mid \sigma(c) \cap \sigma(\mathbf{C}_{2\mathrm{A}} \setminus \mathbf{C}_m) \neq \varnothing \,\};$$
$$\mathbf{C}'_{2\mathrm{A}} = \{\, c \in \mathbf{C}_2 \mid \sigma(c) \cap \sigma(\mathbf{C}_{1\mathrm{A}} \setminus \mathbf{C}_m) \neq \varnothing \,\}.$$

Let's introduce a function $\sigma_\mathrm{P} : \mathcal{P}(\mathbf{C}_\mathrm{P}^\mathrm{U}) \cup \mathbf{C}_\mathrm{P}^\mathrm{U} \rightarrow \mathcal{P}(\mathbf{S})$ which calculates the signature of prefix changes which is just a set of all prefix names and prefix values:

$$\sigma_\mathrm{P}(\mathrm{AddPrefix}(p_n, p_i)) = \{\, p_n, p_i \,\};$$
$$\sigma_\mathrm{P}(\mathrm{RemovePrefix}(p_n, p_i)) = \{\, p_n, p_i \,\};$$
$$\sigma_\mathrm{P}(\mathrm{ModifyPrefix}(p_n, p_i, p_i')) = \{\, p_n, p_i, p_i' \};$$
$$\sigma_\mathrm{P}(\mathrm{RenamePrefix}(p_n, p_i, p_n')) = \{\, p_n, p_i, p_n' \};$$
$$\sigma_\mathrm{P}(\{c_1, c_2, \ldots, c_n\}) = \sigma_\mathrm{P}(c_1) \cup \sigma_\mathrm{P}(c_2) \cup \ldots \cup \sigma_\mathrm{P}(c_n).$$

Conflicting prefix changes are non-matching changes made by different users and having the same prefix name or the same prefix value:

$$\mathbf{C}'_{1\mathrm{P}} = \{\, c \in \mathbf{C}_1 \mid \sigma_\mathrm{P}(c) \cap \sigma_\mathrm{P}(\mathbf{C}_{2\mathrm{P}} \setminus \mathbf{C}_m) \neq \varnothing \,\};$$
$$\mathbf{C}'_{2\mathrm{P}} = \{\, c \in \mathbf{C}_2 \mid \sigma_\mathrm{P}(c) \cap \sigma_\mathrm{P}(\mathbf{C}_{1\mathrm{P}} \setminus \mathbf{C}_m) \neq \varnothing \,\}.$$

4 Architecture and Workflow

The architecture of the collaborative distributed ontology development system is shown in Figure 2.

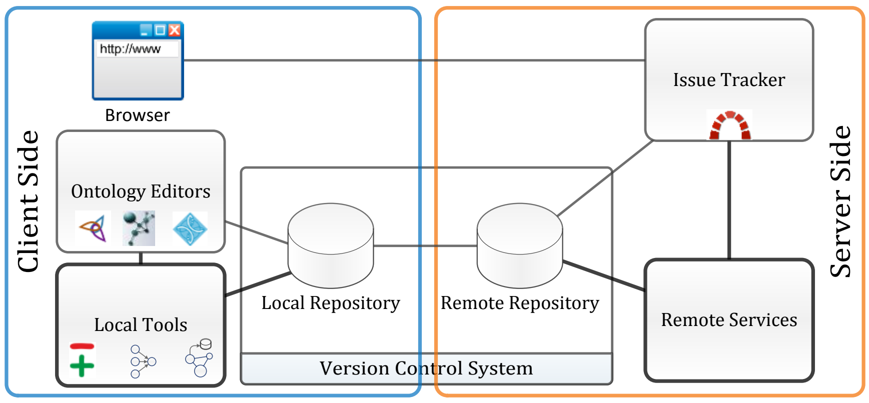


Fig. 2. Implementation architecture

When an ontology developer creates the initial version of an ontology using an ontology editor, he puts it to the local repository of the version control system, and then he publishes (pushes) it to the remote repository. After publishing any developer can get (pull) this version to his local repository and work on it using any ontology editor. During this work, he can compare his version to the initial one using the local diff tool. After changing an ontology, the developer commits his changes to his local repository and then pushed them back to the remote repository in order to make them available for other developers. It is worth noting that it is possible to commit changes (that is create new versions of ontologies) even without network access, for example during a trip or network troubles. Information about each new version is also available through the issue tracker, which can be accessed with a web browser. Remote services perform comparing of adjacent versions and create an index of all entities, which occur in ontologies using the algorithms described above. This index is used to display change log of a specific entity on a special page in the issue tracker where it is easy to see when, by whom and in what ontology the entity was introduced, modified or abolished.

In case when two developers try to push their changes to the remote repository after working on the same ontology, the second developer has to perform the three-way merge operation using the local merge tool in order to ensure the integrity of the ontology. If changes of both developers do not affect the same entities, or if their changes on the same entity are the same, the merge operation is done automatically. From the other hand, if there are conflicting changes, the second developer has to resolve conflicts using the graphical user interface of the local merge tool and an ontology editor (in complex cases).

5 Implementation

The diff and merge tools are implemented in Java using OWL API[2] and are available for download at GitHub[3]. Git or Mercurial can be used as a version control system. Any ontology editor with capability of editing RDF/XML, OWL/XML, OWL Functional Syntax, Manchester OWL Syntax or Turtle can be used with the tools.

5.1 The Diff Tool: owl2diff

The main purpose of the tool is finding changes $\delta(v_1,v_2)$ between two versions v_1 and v_2. It can also perform a shallow analysis of changes in order to find which entities were added, modified, or removed. It takes into account ontology format changes and import changes. It takes two OWL 2 files as an input and produces a change set in form of text, which is close to OWL Functional Syntax. Here is the EBNF grammar of the change set:

[2] http://owlapi.sourceforge.net/

[3] https://github.com/utapyngo/owl2vcs

```
changeset
         ::= prefix*
             setOntologyFormat? setOntologyId?
             prefixChange* importChange*
             annotationChange* axiomChange*
prefix   ::= prefixName '=' fullIRI
setOntologyFormat
         ::= '* OntologyFormat' '(' quotedString ')'
setOntologyId
         ::= '+ ' ontologyIDStatement
           | '- ' ontologyIDStatement
           | '* ' ontologyIDStatement
ontologyIDStatement
         ::= 'OntologyID' '(' oid ')'
oid      ::= fullIRI fullIRI?
prefixChange
         ::= '+ Prefix' '(' prefix ')'
           | '- Prefix' '(' prefix ')'
           | '* Prefix' '(' prefix fullIRI ')'
           | '# Prefix' '(' prefixName prefix ')'
importChange
         ::= '+ ' importsDeclaration
           | '- ' importsDeclaration
importDeclaration
         ::= 'Import' '(' iri ')'
annotationChange
         ::= '+ ' annotation
           | '- ' annotation
axiomChange
         ::= '+ ' axiom
           | '- ' axiom

iri      ::= fullIRI | abbreviatedIRI
literal  ::= quotedString ( '^^' iri | languageTag )?
```

The prefixName, fullIRI, abbreviatedIRI, quotedString, languageTag, annotation and axiom elements are described in OWL 2 Structural Specification [8]. It can be seen from the grammar that change set consists of prefixes used to abbreviate entities, ontology format change, ontology identifier change, prefix changes, import changes, ontology annotation changes, and axiom changes. Each addition of a new axiom, an ontology annotation, an import or a prefix is denoted by the "+" sign, whereas removal is denoted by the "–" sign. Modification of a prefix, an ontology identifier, or an ontology format is denoted by the "*" sign. Renaming of a prefix is denoted by the "#" sign. Here is an example of a change set produced by the owl2diff tool:

```
owl:=<http://www.w3.org/2002/07/owl#>
:=<http://www.co-ode.org/ontologies/pizza/pizza.owl#>
pizza:=<http://www.co-
ode.org/ontologies/pizza/pizza.owl#>

* OntologyFormat("OWL Functional Syntax")
- Prefix(dc:=<http://purl.org/dc/elements/1.1/>)
- DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy )
+ Declaration(NamedIndividual(pizza:Russia))
+ DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy pizza:Russia )
+ ClassAssertion(owl:Thing pizza:Russia)
+ ClassAssertion(pizza:Country pizza:Russia)
```

The first three lines declare the prefixes used in subsequent changes. The first change is the change of ontology format to "OWL Functional Syntax". The following changes are removal of the "dc" prefix, removal of the "DifferentIndividuals" axiom, and addition of four axioms related to adding the "Russia" country to the ontology.

Sometimes it may be useful to see the short representation of the changes (when there are many of them) – just entities affected by the changes. Here is the short representation of the changes shown above produced by owl2diff with the "-e" option:

```
+++ NamedIndividual: pizza:Russia
*** Class: owl:Thing
*** NamedIndividual: pizza:France
*** NamedIndividual: pizza:England
*** NamedIndividual: pizza:Italy
*** NamedIndividual: pizza:Germany
*** Class: pizza:Country
*** NamedIndividual: pizza:America
```

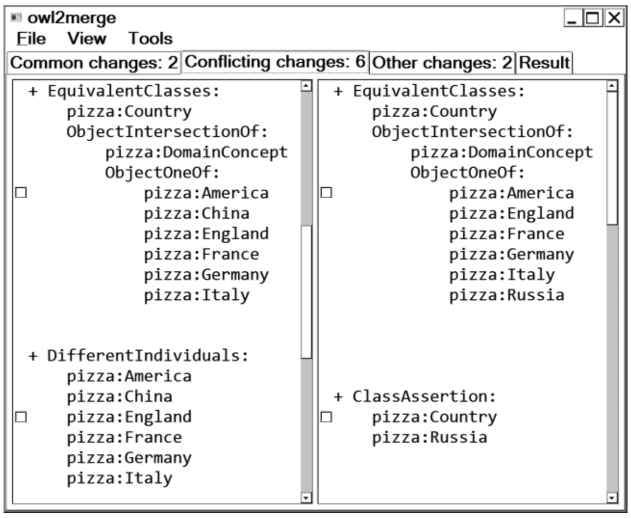
In order to create change logs for each entity, a detailed representation of changes is required. Invoked with the "-c -e" options, the tool outputs all related changes for each entity:

```
+++ NamedIndividual: pizza:Russia
+ Declaration(NamedIndividual(pizza:Russia))
+ DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy pizza:Russia )
+ ClassAssertion(owl:Thing pizza:Russia)
+ ClassAssertion(pizza:Country pizza:Russia)

*** Class: owl:Thing
+ ClassAssertion(owl:Thing pizza:Russia)
```

```
*** NamedIndividual: pizza:France
- DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy )
+ DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy pizza:Russia )
…
*** Class: pizza:Country
+ ClassAssertion(pizza:Country pizza:Russia)

*** NamedIndividual: pizza:America
- DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy )
+ DifferentIndividuals(pizza:America pizza:England
pizza:France pizza:Germany pizza:Italy pizza:Russia )
```

5.2 The Three-Way Merge Tool: owl2merge

The main purpose of this tool is helping the user to resolve conflicts and perform the three-way merge of ontologies. The tool takes three ontology versions as an input (v_0, v_1, v_2) and outputs the resulting ontology v_3, which contains changes made by both users. It uses the same grammar as the diff tool to represent the changes. The graphical user interface of the three-way merge tool is shown in Figure 3. The user interface contains four tabs, which display common (matching) changes, conflicting changes, other changes and resulting changes. By default, all common and non-conflicting changes are selected, and conflicts are to be resolved by the user. If there are no conflicts, changes can be applied automatically without interfering with the user.



```
+ EquivalentClasses:
    pizza:Country
    ObjectIntersectionOf:
        pizza:DomainConcept
        ObjectOneOf:
□           pizza:America
            pizza:China
            pizza:England
            pizza:France
            pizza:Germany
            pizza:Italy

+ DifferentIndividuals:
    pizza:America
    pizza:China
□   pizza:England
    pizza:France
    pizza:Germany
    pizza:Italy
```

```
+ EquivalentClasses:
    pizza:Country
    ObjectIntersectionOf:
        pizza:DomainConcept
        ObjectOneOf:
□           pizza:America
            pizza:England
            pizza:France
            pizza:Germany
            pizza:Italy
            pizza:Russia

+ ClassAssertion:
□   pizza:Country
    pizza:Russia
```

Fig. 3. Conflicting changes tab of the main window of the three-way merge tool

6 Conclusion and Future Work

With owl2vcs, we have local diff and merge tools for web ontologies. It allows us to use any software for editing ontologies (including text editors). If developers are used to a specific ontology editor, they do not have to learn another one. This approach does not restrict ontology language expressiveness. Being based on OWL API, the tools support all OWL constructs including SWRL rules. The solution takes into account peculiarities of ontology storing and allows tracking changes of ontology format. The proposed algorithms are simple and easy to maintain.

However, it would be nice to see the difference between ontologies in the issue tracker using just a web browser. One of the most extensible issue trackers is Redmine[4]. Our next goal is to develop a plugin for Redmine to allow comparing ontologies and analyzing changes at the server side using the algorithms described in this paper.

References

1. Carroll, J. Matching RDF Graphs. (2001), `http://www.hpl.hp.com/techreports/2001/HPL-2001-293.pdf`
2. DiffAndPatch, `http://www.w3.org/wiki/DiffAndPatch`
3. Luczak-Rösch, M., Coskun, G., Paschke, A., Rothe, M., Tolksdorf, R.: Svont – version control of OWL ontologies on the concept level. In: Fähnrich, K.P., Franczyk B. (eds.) Informatik 2010. Service Science – Neue Perspektiven für die Informatik. Band 2. LNI 176, Gl 2010, pp. 79–84. Gesellschaft für Informatik, Bonn (2010)
4. Drummond, N.: OWL Patch (2011), `http://owl.cs.manchester.ac.uk/patch/`
5. Gonçalves, R.S., Parsia, B., Sattler, U.: Ecco: A Hybrid Diff Tool for OWL 2 ontologies. In: Klinov, P., Horridge, M. (eds.) OWLED 2012: OWL Experienced and Directions Workshop, Heraklion, Crete, Greece (2012)
6. Kremen, P., Smid, M., Kouba, Z: OWLDiff: A Practical Tool for Comparison and Merge of OWL Ontologies. In: DEXA 2011, pp. 229-233. IEEE Press (2011)
7. Redmond, T., Noy, N.: Computing the Changes Between Ontologies. In: Novacek, V., Huang, Z., Groza, T. (eds.) EvoDyn 2011: Joint Workshop on Knowledge Evolution and Ontology Dynamics, Bonn, Germany (2011)
8. Motik, B., Patel, P.F., Parsia, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax: World Wide Web Consortium (2009), `http://www.w3.org/TR/owl2-syntax/`

[4] http://www.redmine.org/