Computing the Changes Between Ontologies

Timothy Redmond and Natasha Noy

The Stanford Center for Biomedical Informatics Research,
tredmond@stanford.edu,noy@stanford.edu
http://bmir.stanford.edu

Abstract. As ontologies evolve, the ability to discover how they have changed over time becomes critical. In the recent past, a handful of useful tools based on the Manchester OWL API have addressed this issue. However, none of these tools took into account how to align entities across ontologies when the names of entities have changed. For the case of name changes, we need a way to discover entity alignments between two ontology versions before we can match up structural differences. In this paper, we describe a highly optimized pluggable difference engine that searches for alignments between entities in different ontology versions and applies those alignments to display the differences in the ontologies. This difference engine is available as a plug-in in the latest Protege 4.2 release and sources are available online.[1] We discuss our experiences applying the tools to a selection of ontologies from the BioPortal [5] ontology repository, including the performance and accuracy of the tool.

Keywords: Evolving Ontologies, Version Control

1 Introduction

In any form of document processing, the people who generate and review the documents need to have some method of obtaining a usable collection of differences between the documents. This need arises regardless of whether the document formats in question are text, Word, software code, or, as in our case, the Web Ontology Language (OWL).

For OWL ontologies, many users have to resort to viewing the OWL syntax rendering in a text editor, and using text-based difference tools to locate changes. This approach may sometimes be convenient due to the universal presence of textual difference tools, but falls far short for several reasons. When OWL content is saved to the same format (e.g., RDF/XML) by two different tools, the two files can differ to such a degree that a textual comparison is meaningless. In addition, even when the RDF/XML is generated by a tool that deliberately tries to make the textual differences readable (e.g., the OWL API [8]), the format in which a difference engine displays these changes does not make it clear

[1] Sources for the difference engine library (http://goo.gl/sQ4MZ) and Protege plug-in (http://goo.gl/yIXsO) are located in the Protege Subversion repository.

to an OWL developer how the OWL content has changed in terms of entities or axioms.

In this paper, we discuss a tool that understands the structures in an OWL ontology and calculates the structural differences between ontologies. Specifically, we address the problem of calculating the differences between an earlier version of an ontology (which we call the *source* ontology) and a newer version of the ontology (which we call the *target* ontology).

The OWL language specification [10] defines OWL in terms of the vocabulary of an ontology (*OWL entities*) and structures in the ontology (*OWL Axioms*). We follow this paradigm by first comparing the entities of two ontologies, and then using the results of this comparison to compare the axioms.

OWL entities make up the named terms in an ontology, including classes, properties, individuals, and data types. The OWL language uses Internationalized Resource Identifiers [1] (IRIs) for the naming of OWL entities. When we compare two versions of an ontology we need to define a mapping between the entities that appear in the source ontology and the entities that appear in the target ontology. There are four types of mappings between entities in source and target ontologies:

- An entity appears in the target ontology but there is no analogue in the source ontology (i.e., an entity was created).
- An entity appears in the source ontology but does not appear in the target ontology (i.e., an entity was deleted).
- An entity appears in both ontologies with the same name.
- An entity that appears in the target ontology corresponds to an entity in the source ontology, but the names of the two entities are different.

We call such a mapping of all the entities in the source and target ontologies **an alignment of entities**. The last case in the list above makes the problem of calculating the structural difference between two ontologies both interesting and challenging. There is no guaranteed way to determine which entity in the source ontology corresponds to which entity in the target ontology when the names of entities have changed. In order to effectively determine this correspondence, we will apply a series of heuristics that examine the ontology for clues based on how the entities appear. We will often refer to the final case a *refactor*. When the name of an entity is changed, all referring axioms must be changed to use the new entity name. This is a very similar notion to a refactor as understood by software code developers. In both cases, the tools (e.g., Eclipse for Java code and Protégé for ontologies) call the operation of renaming an entity across multiple documents a "refactor" operation.

The alignment of the OWL entities, in turn, drives the alignment of OWL axioms in source and target ontologies. As with OWL entities, there are four types of mappings between OWL axioms in source and target ontologies:

- An axiom in the source ontology *matches* an axiom in the target ontology if the alignment of the OWL entities maps the structures in the source ontology axiom to the axioms in the target ontology axiom.

- An axiom in the source ontology is *removed* if it does not match any axioms in the target ontology.
- An axiom in the target ontology is *added* if it does not match any axioms in the source ontology.
- An axiom in the target ontology is *changed* if the axiom can be shown to correspond to a different axiom that exists in the source ontology.

In the rest of this paper, we describe the implementation and behavior of a difference engine that we built to calculate the structural difference of two ontologies. We had the following requirements in developing the difference engine:

- calculation of structural differences between ontologies,
- ability to handle refactor operations,
- run fast, perhaps favouring run time performance over memory consumption,
- pluggable, in order to allow users to configure their own alignment and explanation algorithms,
- run either within or outside the Protégé editing environment.

We built a difference engine to meet these requirements, and then applied this tool to several of the ontologies in the NCBO BioPortal [5]. In this paper, we describe how we approached the problem and the results that we achieved.

2 Related Work

Much of the inspiration for this work was derived from a study of the PromptDiff tool [6,7]. Before the OWL 2 specification was released, PromptDiff was the premier tool used by ontology authors who wanted to compare different versions of an ontology. PromptDiff compares different versions of large ontologies, detects when ontology entities have been refactored, and presents the differences between the ontologies. It is pluggable, allowing developers to add custom extensions to the alignment process. Evaluations reported by the PromptDiff authors [6] cite high levels of accuracy and reliability.

The National Cancer Institute (NCI) has been a prototypical user of the PromptDiff tool [13]. A team at NCI develops *NCI Thesaurus* [11], a large-scale biomedical ontology for representing terms relevant to diagnosis and treatment of cancer. There are several developers who contribute to the NCI Thesaurus. At the end of each month, the administrators must examine the changes that these developers have performed, in order to provide quality control before publishing a new production version of the NCI Thesaurus [13]. Thus, the administrators must be able to determine what changes the developers made to the ontology in order to decide whether these changes should be accepted or rejected. NCI uses PromptDiff for this task, running it to compare the contents of the NCI Thesaurus at the end of the editing cycle with the baseline version from the beginning of the cycle. A lead editor at NCI then looks through the changes that were made to the ontology and decides which of those changes to accept or reject.

Unfortunately, PromptDiff has not kept up with the changes to the OWL specification. It is still using an older API for accessing OWL ontologies and this API is not able to handle OWL 2 ontologies. The difference engine that we describe in this paper was inspired by PromptDiff and was born out of the need to support OWL 2. In some ways, these tools appear very different. The difference engine that we describe here uses the Manchester OWL API [8] which takes a very structural view of an ontology. PromptDiff uses a "frame-like" API where an ontology is viewed as a container of property values for named and anonymous entities. At first glance this difference seems quite significant, and indeed, the explanation phase of the difference engine has no analogue in PromptDiff and appears to be a consequence of this differing point of view. However, many of the heuristics that we use during the alignment phase of the difference engine were ported directly from algorithms used by PromptDiff. In addition, the results and explanations of the alignments and differences generated by the two tools are often clearly related to one another. In fact, part of the future work is to port more of the algorithms used by PromptDiff to the difference engine.

ContentCVS [3] is a system based on concurrent versioning, where users check out versions of an ontology and then commit their changes. Different users may check out a copy of a common ontology and make conflicting changes to the ontology. Therefore, ContentCVS provides a framework for concurrent versioning of ontologies, which includes version comparison, conflict detection, and conflict resolution. ContentCVS takes a structural approach to calculating the difference between two ontologies. This approach is similar in spirit to the structural approach that we describe here. In addition, ContentCVS considers issues involving unwanted entailments that occur when multiple users attempt to merge their changes to a baseline ontology. In contrast, our difference engine does not address the problem of merging and detecting conflicts. These are important problems that we have not yet addressed in our tool. But the primary advantage of our difference engine over ContentCVS is that our tool is able to detect many refactor operations.

RDF Deltas [14] addresses the problem of expressing the difference between two RDF graphs using change operations that include side-effects. Including the side-effects as part of the change operations allows the size of the change sets to be minimized. This is very useful in a Semantic Web environment, effectively reducing the bandwidth needed to transmit a collection of changes between two graphs.

The difference engine deviates from RDF Deltas in that we work exclusively with OWL ontologies, RDF Deltas does not deal with refactors, and the RDF Deltas approach involves the use of inference.

The OWLDiff tool [9] is a powerful tool for finding differences between ontologies. It addresses the problem of calculating a difference in the sets of axioms that can be entailed in a source and target ontology. The OWLDiff tool has two modes: a basic ontology comparison and a CEX logical comparison. In the basic ontology comparison, OWLDiff calculates the set difference of the axioms in the

two ontologies and then filters this difference by an analysis of what axioms are entailed by the two ontologies. The CEX logical comparison employs a more complicated algorithm, which uses the fact that in a formally precisely defined sense, the difference between the axioms entailed by the two ontologies can be calculated [4].

The OWLDiff work is distinct from ours in two ways. First, our approach is confined to a purely structural difference. We do not address the problem of entailment. We have focused on the case where the ontology developer wants to understand the set of edits made to an ontology. If, for instance, an axiom was removed because it was already inferred, our tool will not indicate this fact. Second, the OWLDiff tool does not address the problem of understanding refactor operations. In many real-world cases, the names of ontology entities change as the ontology evolves. In some cases, our algorithm is able to apply heuristics that detect these name changes and map axioms in the source ontology to axioms in the target ontology based on these refactor operations.

3 Approach

Our approach to calculating the difference between two ontologies has two phases: the *alignment* phase and the *explanation* phase. In the alignment phase, we determine the difference between the signatures of the two ontologies and calculate any refactors that occurred when the ontology was changed. The explanation phase reorganizes the axiom changes into a format that is more readable to a human and then attempts to represent these changes in the manner that would be the most readable. Both of these phases are pluggable; the body of work is performed by running a collection of algorithms in sequence until no further progress can be made.

3.1 The Alignment Phase

The goal of the alignment phase of the difference algorithm is to generate a raw low level alignment of the source ontology to the target ontology. It does not address issues of how this information should be presented to the end user and it only identifies axioms that have been added or removed; it does not detect when axioms have changed. As it aligns the entities in the signature of the source ontology with entities in the signature of the target ontology, the alignment tool incrementally keeps track of which axioms from the source ontology map to which axioms from the target ontology.

Since we consider only structural differences between OWL ontologies, the case where there are no refactor operations is not particularly interesting. Essentially, the mapping is determined by a set differences of the two signature and axiom sets. However, detecting when the name of an entity changes is a more difficult task and we must rely on heuristics to find the changes that have been made.

In order to find these mappings, the alignment phase repeatedly runs a series of pluggable heuristic algorithms in sequence until the algorithms are no longer making any progress. Each algorithm builds upon the work of previous algorithms to find and add alignments of OWL entities to the pool of discovered alignments. As entity alignments are found, alignments of OWL axioms are incrementally calculated and this information is fed back into the alignment algorithms. Once found, alignments are never removed or altered. As the alignment process progresses, the problem of finding alignments becomes easier because the portion of the ontology that is not aligned becomes smaller and smaller. Therefore, if possible we run the slower algorithms later in the process so that they have a smaller section of the unaligned ontology to take into consideration. In addition, we run less accurate algorithms later in the sequence so that the more accurate algorithms get first pass at finding any alignments. For these reasons, we assign each algorithm a priority that determines when it should run.

To give a flavor of how the alignment works we will give a simple but illustrative example from the Biomedical Resource Ontology (BRO) [12]. We compared version 2.6 against version 3.2.1; both versions can be found in BioPortal [5]. The later version has 486 classes. The contrast between the two versions was a moderately simple diff involving 147 new entities, 58 deleted entities, 83 refactored entities, and 309 entities that were modified but did not fall into the previous categories.

Consider, for example, the following alignment that was identified by our difference engine (Figure 1): the class "Symbolic and Analytical Resource", from version 2.6, was aligned with the class "Symbolic and Analytic Model" from version 3.2 of the ontology. Finding this alignment involved two steps. First ,the difference engine needed to align the class "Models with closed form solutions" from the source ontology with the class "Model with closed form solutions" from the target. This alignment is a typical change that the BRO authors performed during this period. A plural form of the ontology concept was changed to a singular form. The algorithm that discovered his change looked at the children of a concept that was already aligned to find out if any of them had similar names. Since the BRO ontology does not use numeric identifiers, making this change involved also changing the name for the concepts.

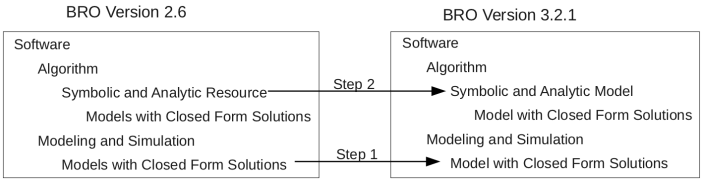

Fig. 1. Alignment in the BRO ontology

Once these classes were aligned, the difference engine had enough information to align the "Symbolic and Analytical Resource" concept. One of our algorithms compares parents and children of concepts from the two ontologies. If a concept in the source ontology has a parent and child that match the parent and child of a concept from the target ontology, then the algorithm adds this as a match. In this case the parent and child of "Symbolic and Analytical Model" in the first ontology were "Algorithm" and "Models with closed form solutions". This parent and child corresponded to a parent and child of "Symbolic and Analytic Model" concept from the second ontology. The parent of "Symbolic and Analytic Model" in the second ontology was the "Algorithm" concept which had an identical name to the "Algorithm" concept from the first ontology. And the child of "Symbolic and Analytic Model" was "Model with closed form solutions".

This example also shows how the alignment algorithms are notified of progress in the alignment of OWL axioms. When the "Models with Closed Form Solutions" is aligned with the "Model with Closed Form Solutions" class, the axiom

```
"Models with Closed Form Solutions"
    SubClassOf "Symbolic and Analytic Resource"
```

moves one step closer to being aligned. The algorithm that looks for matching parents and children is then notified of this movement and this is what tells this algorithm to examine the "Symbolic and Analytic Resource" class in more detail.

Many of the algorithms used by the Protégé 3 PromptDiff engine can be adapted for use in our difference engine. Thus far, we have not implemented very many of these algorithms but we already have enough that we are finding interesting alignments. The current set of algorithms included in the main OWL difference engine are

- *Align by IRI:* This is a highly reliable algorithm that will align entities from the ontologies being compared if they have the same IRI and type. It is very much expected that if two ontologies refer to a concept with the same IRI, then the two concepts are identical. For this reason, the Align by IRI algorithm is run first.
- *Align by rendering:* Many ontologies have some notion of how entities should be rendered. In a lot of cases the rendering is taken from the rdfs:label annotation property. In these cases, the difference engine can match up concepts that have the same rendering even if they have different IRIs. This algorithm is substantially less reliable than the algorithm that aligns by IRI because it is not uncommon, even in a single ontology, for two distinct concepts to have the same rendering.
- *Align by IRI fragment:* In some cases, the name of an entity changes only because the namespace changed. In these cases, we want to align entities that differ only by their namespace.
- *Align siblings with similar renderings:* This algorithm will align siblings of a matched class that have very similar renderings. This type of refactor is very common when an author corrects a typo or misspelling . Also, this algorithm

will often catch the case when modelers change all of their class names from plural to singular.
- *Align entities with aligned parent and child:* If a potentially aligned pair of entities in two ontologies have a matching parent and child we can guess that the entities should be aligned. This is a powerful algorithm which finds some otherwise difficult to match items. Occasionally it finds questionable, yet interesting matches such as the match of "non computational service" in version 2.6.2 of the BRO ontology with "people resource" in version 3.2.
- *Align lone matching sibling:* This algorithm will align entities between two ontologies if all the siblings of the entities in question have been matched. This algorithm currently finds a number of very good matches, but occasionally locates outrageous ones that are clearly wrong. We are still experimenting with the best settings for this algorithm.

With the exception of the last algorithm, these algorithms have been very reliable in practice.

3.2 The Explanation Phase

Running the alignment phase of the algorithm will generate a large amount of information about the relationship between two ontologies. However, this information is not organized for human consumption and is hard to understand without some restructuring. In order to describe our explanation phase, we start by giving example of a typical explanation output and then describe the processing that has taken place to put the raw alignment data into that form.

Consider the difference between versions 2.6.2 and 3.2 of the BRO ontology. The alignment phase of the difference engine will generate a list of entity alignments and an unsorted list of axioms added and removed. In particular, after digging through these results we would find the following entity alignment:

```
Surgical_Procedure -> Surgical_Procedure
```

and hidden among the added and removed axioms we would find the following:

```
Removed:  Surgical_Procedure SubClassOf Novel_Therapeutics
Added:    Surgical_Procedure SubClassOf Therapeutics
Added:    Surgical_Procedure prefLabel "Surgical Procedure"^^string
```

In contrast, the output section of the explanation phase that refers to Surgical_Procedure looks as follows:

```
Renamed and Modified:
  Surgical_Procedure -> Surgical_Procedure
-----------------------------------------------------
Superclass changed:
 Surgical_Procedure SubClassOf Novel_Therapeutics
     changed to
 Surgical_Procedure SubClassOf Therapeutics
 Added:  Surgical_Procedure prefLabel "Surgical Procedure"^^string
-----------------------------------------------------
```

The first two lines of this report indicate that the Surgical_Procedure class has changed names. Indeed in the earlier 2.6.2 version of the ontology, the full name for Surgical Procedure was:

```
http://bioontology.org/ontologies/biositemap.owl#Surgical_Procedure
```

but in the later 3.2 version of the ontology, the full name has become:

```
http://bioontology.org/ontologies/Activity.owl#Surgical_Procedure.
```

Thus far there is no difference between the information being provided here and the information that is available at the end of the alignment phase. However, in the changes listed underneath the alignment of the Surgical_Procedure class there are two significant enhancements to the information that is present in the alignment phase. First, the changes being listed are exactly those changes that are relevant to the entity in question. Thus, in the Surgical_Procedure section we list precisely those axioms describing how the superclass of Surgical_Procedure has changed and how the annotation properties of Surgical_Procedure have changed. This organization was not generated during the alignment phase; the alignment phase result just had a list of axioms added and removed.

Second, the changes that are presented underneath the Surgical_Procedure heading are not confined to simply axiom added and axiom removed. The explanation phase understands that the best way to present the difference given by

```
Axiom Removed: Surgical_Procedure SubClassOf Novel_Therapeutics
Axiom Added:   Surgical_Procedure SubClassOf Therapeutics
```

is to represent this information as a change to the super class of Surgical_Procedure.

This last change is done by a series of pluggable algorithms that detect patterns among changes to axioms and determines the best way to present those patterns to a user. In this case, the work was done by an algorithm called "identify changed superclass." This routine looks for patterns of the form:

```
Axiom Removed:  X SubClassOf Y
Axiom Added:    X SubClassOf Z
```

where X, Y, and Z are named classes. In this case, it will present the change as a change to the superclass of X. It understands that it does not know how to interpret the case where more than one class has been added or removed. For instance, if it finds the following:

```
Axiom Removed:  X SubClassOf Y
Axiom Added:    X SubClassOf Z
Axiom Added:    X SubClassOf T
```

where X, Y, Z, and T are named classes, it will leave the pattern alone.

The fact that the algorithms are pluggable makes it possible to customize the output of the explanation phase to a particular set of ontology development patterns. For example, in the case of NCI Thesaurus, there are annotation properties that indicate that a merge operation took place. An NCI merge operation is performed on two classes, X and Y. All the axioms that describe the Y class are structurally altered so that they talk about X instead of Y. The Y class is then deprecated and it is no longer referenced by any axioms.

To make the difference between two versions of the NCI Thesaurus understandable to the NCI editors, we added an algorithm to "identify merged concepts." This algorithm can present the changes made to X as a "merge with Y" operation. It can present the changes made to Y as a "deprecated because of merge" operation. Finally, it can identify axioms that have been structurally modified because of the merge.

4 Early Tests of the Prototype

We tested the prototype implementation of the difference engine on the NCI Thesaurus and on several ontologies in BioPortal. The primary goals of these tests was to get an initial assessment of the accuracy of the difference engine heuristics and of how the tool performs. We had access to several versions of the NCI Thesaurus and chose four version comparisons. We also tested twenty five of the ontologies from BioPortal. Here we report our findings on the Chemical entities of biological interest (ChEBI) ontology because we had the good fortune to be able to accurately assess the accuracy of our algorithms in this case.

4.1 Performance of the difference engine

Our primary interest in the difference of the NCI Thesaurus is to determine how fast the tool works. The ontology design patterns used by the NCI Thesaurus developers rule out the possibility of refactor operations so we expected that the differences generated by the tool would be relatively simple. However, the performance of initialization of the difference engine and the first few alignment algorithms does not depend on whether or not the ontology will have many refactors. Thus, measuring performance on an ontology with over 87 thousand classes is a useful test. We also paid attention to the performance of the difference engine on the ontologies from BioPortal, though these ontologies tended to be much smaller. The biggest ontology we tested on the BioPortal was the ChEBI ontology which had over 29 thousand classes.

We did our tests on a 2.66 GHz 64-bit quad core Intel machine with 8GB of memory. The difference engine proved to be fast. In all the tested cases the time to calculate the difference between an already parsed ontology against an unparsed ontology was dominated by the time required to parse the second ontology. For example, in comparing version 10.10a against 11.01e, the load of the 10.10a version of the NCI Thesaurus took 56 seconds and the calculation of the differences took 12 seconds. In the case of the ChEBI ontology, parsing the baseline ChEBI ontology took 10 seconds and the difference calculation took 5 seconds. The ChEBI ontology was one of the larger differences that we studied. The difference between versions 1.42 and 1.82 of the ontology, as calculated by the difference engine, involved 10,177 entities created, 121 entities deleted, 97 entities renamed, and 15,509 entities modified.

4.2 Accuracy of the difference engine

We used the ChEBI ontology to evaluate the accuracy of the difference engine. The developers of the ChEBI ontology keep careful track of all the refactors that they perform. They use a numeric identifier naming scheme. When they change the name of an entity, they add an alt_id annotation property that indicates the previous identifier that they used for the entity. The evidence that we have, suggests that they were very

consistent in adding the property. Because our difference engine is (currently) oblivious to the use of the alt_id, we can use the alt_id to verify the accuracy of the refactors that the difference engine finds.

As we mentioned earlier, we observed that our algorithm that aligns lone unmatched siblings is prone to providing false positive matches. Thus, we ran our evaluation for both configurations of the difference engine: with and without this algorithm.

Out of 97 refactors that the difference engine found with "Match lone siblings" turned off, between versions 1.42 and 1.82, we found

- 92 alignments where the difference engine and the alt_id annotation properties were in perfect agreement.
- 2 alignments where we are confident that the difference engine was correct but there is no alt_id annotation property.
- 3 alignments where the difference engine was wrong.

This data suggests that the alt_id mechanism is consistently used in the ChEBI development process (92/94 = 98% of the time). It also suggests that we have a good false positive rate (3/97 = 3.1% false positives).

The false negative result is harder to interpret as a percentage, though it is clear that this result is not as good as the false positive rate. We found that the alt_id annotation indicated 36 alignments between the source and target ontology that the difference engine did not find. But as we work on improving our false negative count we will have to be careful not to increase the false positive result by too much.

Enabling the "Match lone siblings" algorithm made the false positive results significantly worse. In this case, the difference engine had 11 false positives out of 109 refactor operations (11/109=10.1% false positive rate). In other words, 2/3 of the alignments found by the "Match lone siblings" algorithm were wrong. We had noticed this pattern in other cases and had examples of bad alignments in both the EDAM and the BRO ontologies.

Finally, we performed a few experiments where we did some smaller differences of the ChEBI ontologies. The BioPortal includes 31 different versions of ChEBI and we were able to run the difference engine on each pair of ontologies. Overall, the false positive rate when we did the diffs incrementally in this way was slightly higher (5% false positive). In some instances, the false positive rate for an individual difference was significantly higher. In particular, two of the differences had a false positive rate of 7%. This happened because, while there was only one false positive alignment, there were only 13 refactors found altogether. But one of the bigger differences that we tested this way had a false positive rate of only 3/373=.8% and all but three of the differences had no false positives.

5 Discussion

In this paper, we have described a heuristic-based tool for finding alignments between versions of OWL 2 ontologies.

Note that while on the surface there may be similarities with the approaches to mapping different ontologies (not versions of the same ontology) [2], we believe that the heuristics we used are applicable largely only in the context of comparing two versions. They rely on the fact that two versions of the same ontology do not differ nearly as much as two ontologies that came from different sources.

Naturally, we would not need to use heuristics to identify the refactor operations, if the tools for ontology development enabled developers to track these refactors and to specify them declaratively in the ontology. We have found, for instance, that the ChEBI ontologies maintain an alt_id annotation property that serves as a pointer to previous versions of a particular concept. For the ChEBI, this property serves as a highly accurate way of aligning entities from two versions of the ontology. In a similar way, the NCI tools used annotation properties to record merge and split operations.

One might think that refactor operations would become less common as ontology developers move towards using "meaningless" identifiers for their concepts. However, in our experiments with the BioPortal ontologies we found refactor operations among many ontologies with "meaningless" identifiers.

Specifically, there are two strategies that ontology developers take regarding the naming of entities. One approach is to make the name of an entity correspond to something meaningful that users can understand. An example of an ontology that takes this approach is BRO. For example, to follow is the IRI for the class "Area of Research" in BRO:

```
http://bioontology.org/ontologies/ResearchArea.owl#Area_of_Research.
```

Because in this approach, an IRI for a class has to change any time a typo or a misspelling is fixed, many ontology-development projects consider it to be a good practice to use meaningless identifiers. In this case, the names for the entities consist of a prefix followed by a numeric part. So for instance the entity representing "sugar" in ChEBI has the following IRI:

```
http://purl.obolibrary.org/obo/chebi_16646.
```

In order to associate this entity with its readable name (sugar), the ChEBI ontology provides the entity with an rdfs:label annotation with the value "sugar". In theory, when an ontology is developed in this manner, entity names will never change and the chebi_16646 will always refer to the concept associated with "sugar."

However, in our experiments, we have found that in several cases, the name of an entity does change even when the ontology is using numeric names for its entities. Generally, this change occurs because the namespace used for the ontology entities has changed. Thus, we can still use the numeric fragment at the end of the full name to guide how the entities in the two ontologies should be aligned. We noticed this pattern in several of the ontologies in BioPortal [5].

Even when numeric identifiers are used and the namespace is not changed, there are occurrences of good matches where the full name of the concept changes. For example, we did a comparison of the ChEBIontology version 41 against version 81. We found that the "1,2-oxazoles" concept from version 41 was changed to be the "isoxazoles" concept in version 81. The ChEBI ontology uses numeric identifier to identify their concepts. But in the first ontology, the name of the "1,2-oxazoles" concept ended with chebi_46813 and the name of the "isoxazoles" concept ended with chebi_55373. The ontologies in the ChEBI group were clearly aware of this refactor because in the new ontology there was an annotation that indicated that the alternative id for the "isoxazoles" concept was "chebi:46813".

6 Conclusion

In this paper we have presented a tool that will do a structural comparison of OWL ontologies. We showed that the algorithm runs reasonably quickly, so that even large ontologies like the NCI Thesaurus can be easily compared.

In addition, the difference engine has demonstrated the ability to detect non-trivial refactor operations. In several of the ontologies from the BioPortal, we found alignments that are not obvious from the perspective of someone who is not a domain expert. For example, in the ChEBI ontology, there are several cases where the difference engine would align a concept with a very long label, e.g. 9-(2,3-dideoxy-beta-D-ribofuranosyl)-1,9-dihydro-6H-purin-6-one) with a different concept with a much shorter label, e.g. didanosine. In these cases, the only way that we could determine that the alignment was indeed valid was to notice that the latest version of the ChEBI ontology includes an alt_id annotation property value that indicates that didanosine has an alternative id of chebi:39738.

As well as being an effective tool, the difference engine is highly flexible. In our work for NCI, we used a different set of alignment and explanation algorithms that were custom-tailored to fit their specific needs. We aligned entities based on the code annotation property as this is guaranteed to be an immutable identifier for entities developed by an NCI editor. We changed the explanation algorithm to detect merge and split operations and to present this information in a way that can be easily understood in the NCI context.

References

1. Duerst, M., Suignard, M.: Internationalized Resource Identifiers (IRIs). http://www.ietf.org/rfc/rfc3987.txt (January 2005), IETF Network Working Group
2. Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Berlin ; New York (2007)
3. Jiménez-Ruiz, E., Grau, B., Horrocks, I., Berlanga, R.: Building ontologies collaboratively using contentcvs. In: Proc. of the International Workshop on Description Logics (DL). vol. 477. Citeseer
4. Konev, B., Walther, D., Wolter, F.: The logical difference problem for description logic terminologies. Automated Reasoning pp. 259–274 (2008)
5. Noy, N., Shah, N., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Montegut, M., Rubin, D., Youn, C., Musen, M.: Bioportal: A web repository for biomedical ontologies and data resources. In: Demo session at 7th International Semantic Web Conference (ISWC 2008). Citeseer (2008)
6. Noy, N., Kunnatur, S., Klein, M., Musen, M.: Tracking changes during ontology evolution. The Semantic Web–ISWC 2004 pp. 259–273 (2004)
7. Noy, N., Musen, M.: Promptdiff: A fixed-point algorithm for comparing ontology versions. In: Proceedings of the National Conference on Artificial Intelligence. pp. 744–750. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2002)
8. OWL API. `http://owlapi.sourceforge.net`
9. OWLdiff Project. `http://krizik.felk.cvut.cz/km/owldiff/`
10. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`

11. Sioutos, N., de Coronado, S., Haber, M., Hartel, F., Shaiu, W., Wright, L.: NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. Journal of Biomedical Informatics 40(1), 30–43 (2007)
12. Tenenbaum, J.D., Whetzel, P.L., Anderson, K., Borromeo, C.D., Dinov, I.D., Gabriel, D., Kirschner, B., Mirel, B., Morris, T., Noy, N., Nyulas, C., Rubenson, D., Saxman, P.R., Singh, H., Whelan, N., Wright, Z., Athey, B.D., Becich, M.J., Ginsburg, G.S., Musen, M.A., Smith, K.A., Tarantal, A.F., Rubin, D.L., Lyster, P.: The biomedical resource ontology (bro) to enable resource discovery in clinical and translational research. Journal of Biomedical Informatics 44, 137–145 (2011)
13. Tudorache, T., Noy, N., Tu, S., Musen, M.: Supporting collaborative ontology development in protégé. The Semantic Web-ISWC 2008 pp. 17–32 (2008)
14. Zeginis, D., Tzitzikas, Y., Christophides, V.: On the foundations of computing deltas between rdf models. The Semantic Web pp. 637–651 (2007)