Tracking Complex Changes During Ontology Evolution

Natalya F. Noy
Stanford Medical Informatics,
Stanford University,
Stanford, CA 94305
noy@smi.stanford.edu

Michel Klein
Vrije University Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
michel.klein@cs.vu.nl

1 The Need and Requirements for Version Comparison

For the Semantic Web to succeed, it will require the development and integration of numerous ontologies. As ontology development becomes a more ubiquitous and collaborative process, support for *ontology versioning* [Klein, 2001; Noy and Klein, 2003] becomes necessary and essential. This support must enable users to compare versions of ontologies and analyze differences between them.

There are several reasons to maintain and compare ontology versions. First, ontologies that support the Semantic Web undergo **regular changes**, just as other artifacts do. Second, as ontologies become larger, **collaborative development** of ontologies becomes common. Ontology designers working in parallel on the same ontology need to maintain and compare different versions, examine the changes that others have performed, and so on. Third, the more expressive languages for the Semantic Web, such as DAML+OIL and OWL, are Description Logic (DL) languages. One can view the task of **comparing the asserted and the inferred subsumption hierarchies** in a DL ontology as a versioning problem: The user needs to see how the classification has changed the hierarchy, where were the classes moved, and so on.

We can reuse some of the approaches from the fields of software versioning and collaborative document processing for ontology versioning, but we must keep in mind one crucial difference: In the case of software code and documents, what is compared are *text files*. For ontologies, we need to compare the *structure* and *semantics* of the ontologies and not their textual serialization.

2 Complex ontology changes

The first step in comparing the structure of ontologies rather than their textual serialization is establishing correspondences between concept definitions in two versions, identifying that a concept A in one version became A' in the other.

Identifying correspondences between concepts in different versions leads directly to the second step: identifying simple changes between versions, such as addition or deletion of concepts, change in concept defintions, and so on.. However, in order to assist users in analyzing and understanding the changes that have occurred from one version to another, we must identify *complex* changes as well: For example, it is more useful to know that a concept was *moved* from one place in the hierarchy to another than to know that it was deleted from one and added to the other.

More specifically, the following are some of the complex changes that we have identified.

Add a subtree: Create a new class and create one or more of its subclasses.

Delete a subtree Delete a class and all its subclasses.

Move a subtree to a different location Move a subtree of classes to a different location in the class hierarchy. This operation is essentially equivalent to changing a superclass of the root of this subtree.

Move a set of sibling classes to a different location Move two or more classes that are siblings in the class hierarchy to the same new location in the class hierarchy (i.e., they remain siblings, but under a different parent).

Create a new abstraction Move a set of siblings down in a class hierarchy, creating a new superclass.

Remove an abstraction Delete a class, moving its subclasses to become subclasses of its superclass.

Split a class Split a class into two or more sibling classes.

Merge classes Merge two or more siblings into a single class.

3 User Interface

We have developed PROMPTDIFF, a tool for tracking changes between ontology versions [Noy and Musen, 2002]. It is a plugin to the Protégé ontology environment [Protege, 2002].

Figure 1 shows how PROMPTDIFF presents the result of comparing two versions of the UNSPSC ontology, which is a standardized hierarchy of products and services that enables users to consistently classify the products and services they buy and sell. User input results in regular updates, consisting, for example, of additions of new products, or re-classifications of existing products. In the PROMPTDIFF result, the classes that were deleted are crossed out, the added classes are underlined, and classes that were renamed or changed are in bold. We use color coding to make the changes even more apparent. The warning icon (⚠) overlayed with the class icon indicates that the subtree rooted at the class has undergone some changes.

Figure 2 shows *complex changes* in these two versions of the UNSPSC ontology: The addition of several classes rooted at $Distribution\_and\_Control\_centers\_and\_accessories$ is in fact a tree addition. The icon at the root of the added

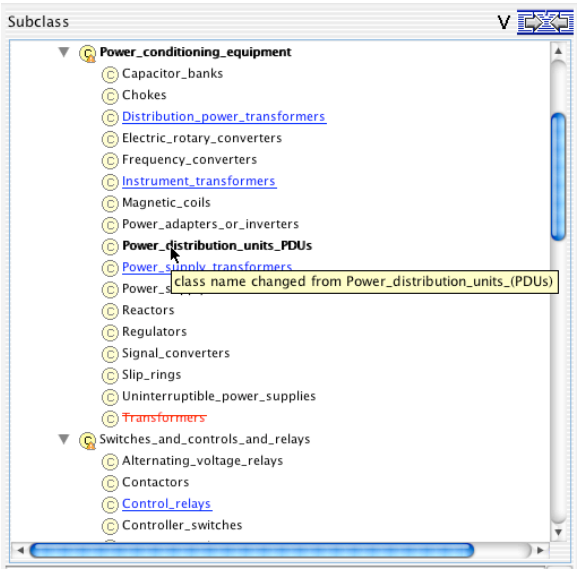

Figure 1: Comparison of two versions of the UNSPSC ontology in PROMPTDIFF. The classes that were deleted are crossed out and the added classes are underlined.

subtree has an overlayed add icon (⊕) indicating that all classes in this subtree have the same status—they were all added in this version. If a whole tree is deleted, an overlayed delete icon (⊖) identified the tree-level operation. The class *Electrical_equipment_and_components_and_supplies* was moved to this location from another position in the tree. The tooltip indicates where it was moved from.

Figure 3 shows the moved class in its old position in the hierarchy: The class appears in grey and the tooltip indicates where the class was moved to.

To summarize, we visualize two types of changes: (1) class-level changes and (2) tree-level changes. For class-level changes, the class-name appearance indicates whether the class was added, deleted, moved to a new location, moved from a different location, or its name or definition has

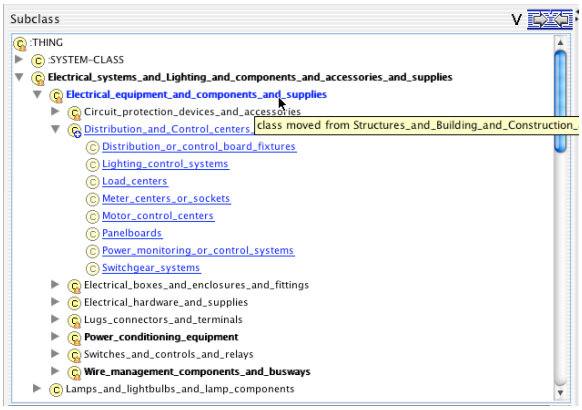

Figure 2: A comparison thats shows a moved class (in bold) and the addition of a subtree.

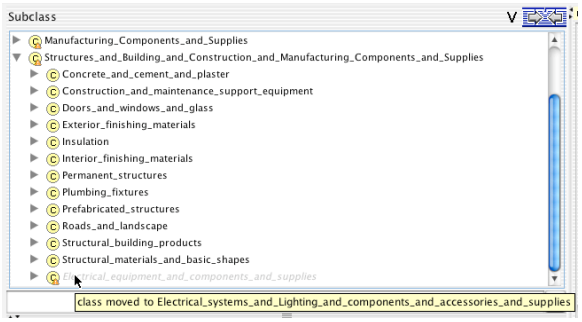

Figure 3: The old position of the moved class (see Figure 2).

changed. If all classes in a subtree have changed in the same way (e.g., were all added or deleted), then the changed icon at the subtree root indicates that the tree-level operation.

4 Outlook

We have presented a tool for examining changes between ontology versions and identified a set of complex changes between ontology versions. Currently, PROMPTDIFF does not display all the changes presented in Section 2, although internally it identifies all of them. We plan to experiment with additional visual metaphors for displaying all complex changes and to evaluate whether using too many different visual clues puts too much of a cognitive load on the user.

Another natural extension of the current tool would be enabling users to accept and reject changes. the default We can also consider using logs of changes if they are available (perhaps grouping together some basic changes in the log into single complex changes) to determine the differences between versions. comparing ontology concepts in likely have

Finally, as we gain more experience with ontology versioning, we will be able to identify more complex changes between versions, and, more important, find automatic ways of determining that such changes have occurred.

Acknowledgments

This research was supported in part by a contract from the U.S. National Cancer Institute.

References

[Klein, 2001] M. Klein. Combining and relating ontologies: an analysis of problems and solutions. In *IJCAI-2001 Workshop on Ontologies and Information Sharing*, pages 53–62, Seattle, WA, 2001.

[Noy and Klein, 2003] Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 5, 2003. in press.

[Noy and Musen, 2002] N. F. Noy and M. A. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, Alberta, 2002.

[Protege, 2002] Protege. The Protégé project, http://protege.stanford.edu, 2002.