Ontology Versioning in an Ontology Management Framework

Natalya F. Noy and Mark A. Musen, *Stanford University*

Ontologies have become ubiquitous in information systems. They constitute the Semantic Web's backbone, facilitate e-commerce, and serve such diverse application fields as bioinformatics and medicine. As ontology development becomes increasingly widespread and collaborative, developers are creating ontologies using different tools and different languages. These ontologies cover unrelated or overlapping domains at different levels of detail and granularity.

A uniform framework for managing ontology versions will help developers work more effectively with existing ontologies and create new ones.

This growth inevitably produces an ontology management problem: ontology developers and users must be able to find and compare existing ontologies, reuse complete ontologies or their parts, maintain different versions, and translate between different formalisms. In short, ontology developers face problems similar to those that software engineers have faced for many years.

A uniform framework, which we present here, helps users manage multiple ontologies by leveraging data and algorithms developed for one tool in another. For example, by using an algorithm we developed for structural evaluation of ontology versions, this framework lets developers compare different ontologies and map similarities and differences among them.

Managing multiple ontologies

Multiple-ontology management includes these tasks:

- *Maintain ontology libraries.* Allow uniform access to ontologies in a library; provide pertinent information about each ontology such as its authors, domain, and documentation; provide search capabilities across all ontologies in a library; and permit browsing of the ontologies themselves.
- *Import and reuse ontologies.* Let users extend and customize ontologies others developed.
- *Translate ontologies from one formalism to another.* Ensure interoperability of ontology development tools by providing translators or import-export mechanisms for ontologies developed using different tools.
- *Support ontology versioning.* Provide mechanisms to store and identify various versions of the same ontology and to highlight differences between them.
- *Specify transformation rules between different ontologies and versions.* Enable transformation of one ontology's instance data to another ontology.
- *Merge ontologies.* Create a new ontology that incorporates information from all given source ontologies.
- *Align and map between ontologies.* Define correspondences between different ontologies' concepts and relations.
- *Extract an ontology's self-contained parts.* Analyze dependencies and let users extract sets of concepts and relations as a subontology.
- *Support inference across multiple ontologies.* Use mappings defined between ontologies to support inference across several ontologies.
- *Support query across multiple ontologies.* Use mappings to support queries to one ontology posed in terms of another.

This list includes only the tasks we face today and will likely grow as more diverse and overlapping ontologies appear. Currently, most researchers treat these tasks as completely independent ones, and the corresponding tools are also independent from one another. Ontology-merging tools (such as Chimaera[1]) have no relation to ontology-mapping tools (such as ONION[2]) or ontology-versioning tools (such as

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

OntoView[3]). Researchers developing formalisms for specifying transformation rules from one ontology version to another don't usually apply these rules to related ontologies that aren't versions of each other.

However, many multiple-ontology management tasks interrelate and have common elements and subtasks, and tools for supporting some tasks can benefit from integration with others. For example, ontology-merging methods we develop to help users find overlap between ontologies can also work for finding differences between ontology versions.[4] In both cases, we have two overlapping ontologies and need to determine a mapping between their elements. When we compare ontologies from different sources, we concentrate on similarities, whereas in version comparison we need to highlight the differences. These processes can be complementary.

In a previous study, we used heuristics similar to those we present here to provide suggestions in interactive ontology merging.[5] Because ontology versioning deals with two versions of the same ontology, we can use the same techniques but require significantly less user input and verification. We concentrate here on the ontology-versioning aspect of multiple-ontology management.

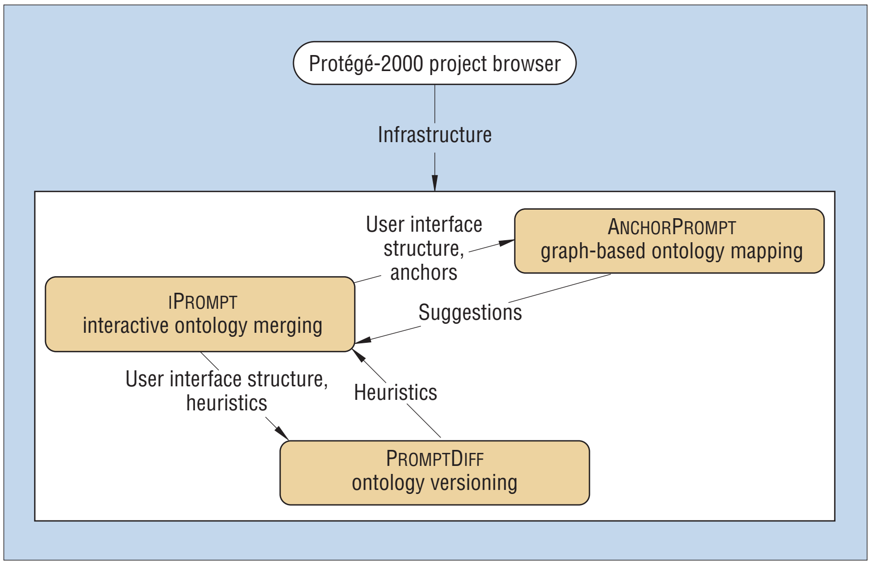

Protégé-2000 project browser
Infrastructure
User interface structure, anchors
ANCHORPROMPT graph-based ontology mapping
IPROMPT interactive ontology merging
Suggestions
User interface structure, heuristics
Heuristics
PROMPTDIFF ontology versioning

Figure 1. The Prompt ontology management infrastructure.

An integrated infrastructure for multiple-ontology management

Consider a set of ontologies. This set can be rather small, such as a set of local ontologies developed by a single user. It can be large, such as an ontology library for an organization or a community, or larger yet, such as all ontologies in the Semantic Web. An ontology set often includes ontologies a user needs to relate to one another—perhaps ontologies from other projects that a user wants to merge to create a single coherent ontology, or different versions of the same ontology that the user needs to analyze. In these cases, the user picks two or more ontologies to work with and looks for overlap between them. We can express the overlap as a set of declarative mapping rules or operational rules that would transform one ontology into the other.

We gain significant advantages from considering these tasks together rather than independently:

- We can leverage algorithms for finding similarities between overlapping ontologies from various sources to find differences between ontology versions. We use different thresholds to decide whether two frames are similar, but the underlying analysis can be the same. Similarly, we can use heuristics discovered when comparing ontology versions to compare different ontologies.
- Because ontology management tasks involve comparing several ontologies, analyzing and understanding semantic relations between their elements can be cognitively difficult. Regardless of whether the ontologies are headed for merging or alignment or the user simply wants to compare them, a uniform interface that shows similarities and differences between ontologies, suggestions for integrating them, and visualization of large-scale ontologies and relations among them can greatly reduce the user's cognitive load.

Figure 1 presents our Prompt ontology management framework. All components are plugins or extensions to the Protégé ontology development environment (http://protege.stanford.edu). Protégé provides an intuitive graphical user interface for ontology development, a rich knowledge model that lets us test our tools with different knowledge-modeling features, and an extensible architecture that provides API access to both the Protégé knowledge bases and its user interface components.

This framework assembles several ontology management tools and provides an infrastructure for other related tools. Its key components include

- *iPrompt*, an interactive ontology-merging tool that helps users merge ontologies by providing suggestions, analyzing conflicts, and suggesting conflict resolution strategies[5]
- *AnchorPrompt*, a graph-based tool for finding related concepts in different ontologies[6] that takes as input pairs of related terms in the source ontologies and analyzes the ontologies' graph structure to find new pairs of related terms
- *PromptDiff*, an ontology-versioning tool we describe later that finds a *structural diff* (that is, determines what has changed) between versions of the same ontology
- *The Protégé project browser*, which provides access to an ontology library, giving users meta-information about ontologies (such as authors, documentation, and modification date), snapshots of top levels of ontologies, and allowing users to search through classes and slots in all ontologies

These tools interact closely. iPrompt provides interface components for other tools that let users browse two ontologies side by side, use different colors for concepts from different ontologies, list pairs of related terms, and so on. iPrompt also provides pairs of related terms to AnchorPrompt. Analysis in AnchorPrompt in turn provides additional suggestions that iPrompt can present to users. PromptDiff uses heuristics we developed in iPrompt to compare ontology versions. We concentrate here on the problem of comparing ontology versions and present an algorithm that automatically finds differences between them.

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

Related Work

Philip Bernstein and his colleagues explored the possibility of creating a uniform view of model management for database systems applications.[1] They viewed a model as a complex structure, such as a relational schema, a UML (Unified Modeling Language) model, an XML document type definition, or a semantic network. They developed a formal framework for mapping between models, using a mapping as a formal structure that contains expressions linking concepts in one model to those in another model. This mapping can then be used for transferring instance data, schema integration, schema merging, and other similar tasks.

Current ontology-versioning research addresses three main issues:

- Identifying ontology versions in distributed environments such as the Semantic Web[2]
- Explicitly specifying change logs between versions[3–5]
- Determining a set of additional ontology changes that each user-specified change incurs[6,7]

However, change logs might not always be available, especially with distributed ontology development. So, our research focuses on developing an automatic way to compare different versions on the basis of the semantics encoded in their structure.

The OntoView ontology version management system[8] also compares source ontologies' structures and identifies change types between versions of the same concept. However, if a concept name changes, OntoView doesn't attempt to determine whether the newly named concept is the same as an old concept. That is, OntoView concentrates on describing differences between concepts that would be in the same row of the PromptDiff table. OntoView does let its users augment a conceptual description of how the concept has changed.

Schema-evolution and versioning research in databases also assumes the availability of a record of changes between versions.[9] Researchers usually identify a canonical set of schema change operations and consider how these operations affect instance data as it migrates from one version to another.[10]

Unlike schema evolution research, database-schema integration automates comparison between schemas originating from different sources. Erhard Rahm and Philip Bernstein surveyed approaches that use linguistic techniques to look for synonyms, machine-learning techniques to propose matches based on instance data, information retrieval techniques to compare attribute information, and so on.[11] In fact, this field potentially can supply many heuristic matchers to the PromptDiff algorithm. However, because none of these algorithms was designed to compare versions of the same schema, only different schemas, it will be interesting to see how well they perform in our case.

One schema integration algorithm does rely on source schemas being similar. In designing the TranScm system for data translation, Tova Milo and Sagit Zohar observed that when translating data from an XML document to an object-oriented database, for example, the underlying schemas are often similar because they describe the same data type.[12] A few explicit rules can thus account for many transformations. TranScm could benefit from many of the simple heuristics we describe here, and PromptDiff could use some TranScm rules as its matchers.

Although semiautomated ontology-mapping research compares disparate ontologies rather than versions of the same ontology, its tools might offer some useful PromptDiff extensions. GLUE uses machine-learning techniques to find matches between classes.[13] AnchorPrompt[14] and the SKAT tool's articulation engine[15] both use similarities in the ontology graph structure to suggest candidate matches. Because they evaluate dif-

Finding structural diffs between ontologies

Ontology developers now face the same problem software engineers began encountering long ago: versioning and evolution. Software code version management tools such as CVS (Concurrent Versions System, www.cvshome.org) have become indispensable for software engineers participating in dynamic collaborative projects. These tools provide a uniform version storage mechanism, the ability to check out particular code segments for editing, an archive of earlier versions, and mechanisms for comparing versions and merging changes and updates.

Like software, ontologies change. These changes can be caused, for example, by domain modifications (that is, our knowledge about the domain or the domain itself) or altered conceptualization (if we introduce new distinctions or eliminate old ones). Furthermore, ontology development in large projects is a dynamic process in which multiple developers participate, releasing subsequent ontology versions. Naturally, collaborative development of dynamic ontologies requires tools similar to software-versioning tools. In fact, ontology developers can use the storage, archival, and check-out mechanisms of tools such as CVS with few changes.

One crucial difference exists, however: comparing software code versions entails simply comparing text files. Program code consists of text documents, and comparing them—the diff process—yields a list of lines that differ in the two versions. This approach doesn't work for comparing ontologies: two ontologies can be exactly the same conceptually but have different text representations. For example, their storage syntax or the order in which they introduce definitions in the text file might differ, or a representation language might use several mechanisms to express the same thing. Text-file comparison thus proves largely useless in comparing ontology versions. The PromptDiff algorithm compares ontology version *structures*, not their text serialization.

We assume the following knowledge model: an ontology has classes, a class hierarchy, instances of classes, slots as first-class objects, slot attachments to class to specify class properties, and facets to specify constraints on slot values. These elements also exist in other representation formalisms such as RDF-S (Resource Description Framework Schema, www.w3c.org/rdf) and OWL (Web Ontology Language, www.w3.org/TR/owl-features), sometimes in a slightly different form. So, our results apply to ontologies defined in these languages as well.

Suppose we're developing an ontology of wines. The first version (see Figure 2a) has a class Wine with three subclasses, Red wine, White wine, and Blush wine. Wine has a slot maker whose values are instances of the class Winery. Red wine has two subclasses, Chianti and Merlot. Figure 2b shows a later version of the same ontology fragment, in which we've changed the name of maker to produced_by and the name of Blush wine to Rosé wine, we added a tannin_level slot to Red wine, and we discovered that Merlot can be white and added another superclass to Merlot.

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

ferent ontologies, these algorithms must be more conservative in their comparisons, requiring more than a simple name and type match between frames to declare their similarity. However, because PROMPTDIFF is easily extensible, we can incorporate these algorithms as new matchers in the fixed-point stage and integrate the results. Furthermore, most algorithms we've mentioned either don't use the semantics of links at all or treat only is-a links as a special case. In the matchers we describe here, we've used the semantics of is-a, instance-of, slot attachment, slot range, and facet attachment links.

References

1. P.A. Bernstein, A.Y. Halevy, and R.A. Pottinger, "Model Management: Managing Complex Information Structures," *SIGMOD Record*, vol. 29, no. 4, Dec. 2000, pp. 55–63.
2. M. Klein, "Combining and Relating Ontologies: An Analysis of Problems and Solutions," *Proc. Workshop Ontologies and Information Sharing at the 17th Int'l Joint Conf. Artificial Intelligence* (IJCAI 2001), 2001, pp. 53–62; http://CEUR-WS.org/Vol-47.
3. J. Heflin and J. Hendler, "Dynamic Ontologies on the Web," *Proc. 17th Nat'l Conf. Artificial Intelligence* (AAAI 2000), AAAI Press, 2000, pp. 443–449.
4. D. Ognyanov and A. Kiryakov, "Tracking Changes in RDF(S) Repositories," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management* (EKAW 2002), LNCS 2473, Springer-Verlag, 2002, pp. 373–378.
5. D.E. Oliver et al., "Representation of Change in Controlled Medical Terminologies," *Artificial Intelligence in Medicine*, vol. 15, Jan. 1999, pp. 53–76.
6. A. Maedche et al., "Managing Multiple Ontologies and Ontology Evolution in Ontologging," *Proc. 17th Conf. Intelligent Information Processing* (IFIP 02), Kluwer, 2002, pp. 51–63.
7. L. Stojanovic et al., "User-Driven Ontology Evolution Management," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management* (EKAW 2002), LNCS 2473, Springer-Verlag, 2002, pp. 285–300.
8. M. Klein et al., "Ontology Versioning and Change Detection on the Web," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management* (EKAW 2002), LNCS 2473, Springer-Verlag, 2002, pp. 197–212.
9. J.F. Roddick, "A Survey of Schema Versioning Issues for Database Systems," *Information and Software Technology*, vol. 37, no. 7, July 1995, pp. 383–393.
10. J. Banerjee et al., "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," *Proc. ACM SIGMOD Conf.*, ACM Press, 1987, pp. 311–322.
11. E. Rahm and P.A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *VLDB J.*, vol. 10, no. 4, Dec. 2001, pp. 334–350.
12. T. Milo and S. Zohar, "Using Schema Matching to Simplify Heterogeneous Data Translation," *Proc. 24th Int'l Conf. Very Large Data Bases* (VLDB 98), Morgan Kaufmann, 1998, pp. 122–133.
13. A. Doan et al., "Learning to Map between Ontologies on the Semantic Web," *Proc. 11th Int'l WWW Conf.*, ACM Press, 2002, pp. 662–673.
14. N.F. Noy and M.A. Musen, "Anchor-PROMPT: Using Non-Local Context for Semantic Matching," *Proc. Workshop Ontologies and Information Sharing at the 17th Int'l Joint Conf. Artificial Intelligence* (IJCAI 2001), 2001, pp. 63–70; http://CEUR-WS.org/Vol-47.
15. P. Mitra, G. Wiederhold, and M. Kersten, "A Graph-Oriented Model for Articulation of Ontology Interdependencies," *Proc. Conf. Extending Database Technology* (EDBT 2000), LNCS 1777, Springer-Verlag,

PROMPTDIFF automatically produces a table (see Figure 3) showing differences between the two versions—similar to the diff between text files, this table presents a *structural diff*. The first two columns show pairs of matching frames from the two ontologies. Given two versions of an ontology O, V_1 and V_2, frames F_1 from V_1 and F_2 from V_2 match if F_1 became F_2. The third column identifies whether the frame has been renamed. The *operation* column shows the user how a frame has changed: whether it was added or deleted, split in two frames, or merged with another frame. We assign a *map* operation to a frame pair if no other operation applies. *Map level* indicates whether the matching frames differ enough to warrant user attention. If the map level is *unchanged*, the user can safely ignore the frames—nothing has changed in their definitions. If two frames are *isomorphic*, their corresponding slots and facet values are images of each other but not necessarily identical. The map level is *changed* if the frames have slots or facet values that aren't images of each other.

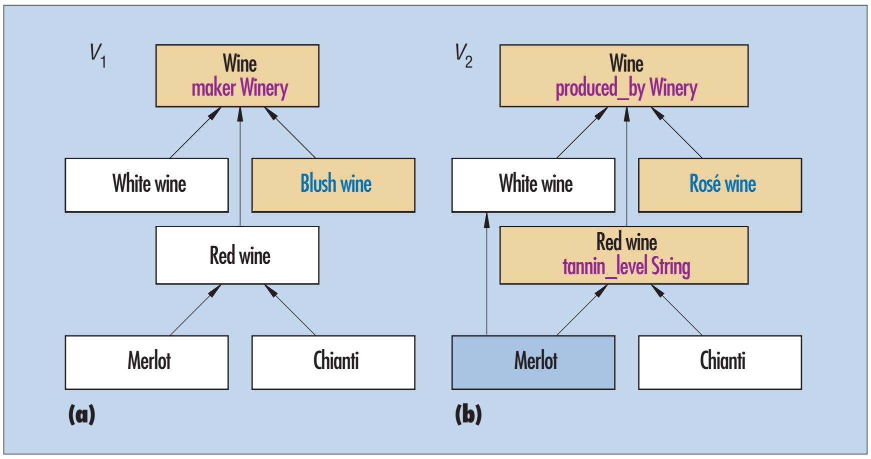
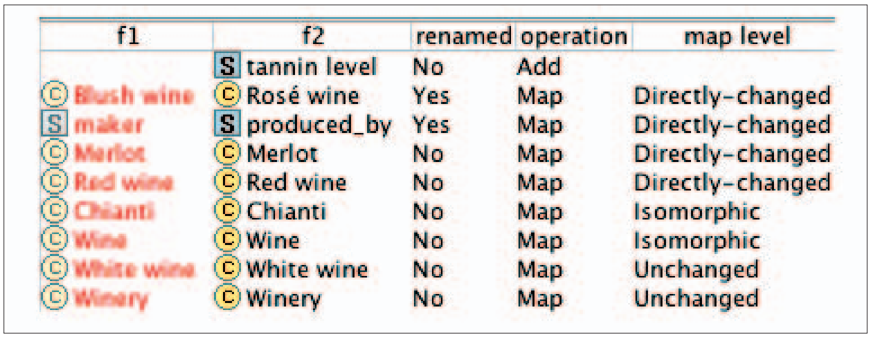


Figure 2. Two versions of a wine ontology. Between the original (a) and this revision (b), changes (indicated by shading) include the slot name for the Wine class, the name of the Blush wine class, a new slot added to the Red wine class, and another superclass for the Merlot class.

In Figure 3, Red wine has changed: it got a new slot. Also, the Chianti class pair is marked as *isomorphic*: the frames themselves haven't changed but the frames they directly reference

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.



f1	f2	renamed	operation	map level
	S tannin level	No	Add	
C Blush wine	C Rosé wine	Yes	Map	Directly-changed
S maker	S produced_by	Yes	Map	Directly-changed
C Merlot	C Merlot	No	Map	Directly-changed
C Red wine	C Red wine	No	Map	Directly-changed
C Chianti	C Chianti	No	Map	Isomorphic
C Wine	C Wine	No	Map	Isomorphic
C White wine	C White wine	No	Map	Unchanged
C Winery	C Winery	No	Map	Unchanged

Figure 3. A PromptDiff table shows differences between the wine ontology versions in Figure 2.

(Red wine) have.

The PromptDiff algorithm has two parts:

1. An extensible set of heuristic matchers
2. A fixed-point algorithm to combine the matchers' results to produce a structural diff between two versions

Each matcher employs a small number of the structural properties in an ontology to produce matches. The fixed-point step invokes the matchers repeatedly, feeding one matcher's results into the others until they produce no more changes in the diff.

We based our approach to automating the comparison on two experimental observations. First, when we compare two versions of the same ontology, a large number of frames remain unchanged. Second, two frames of the same type (for example, both classes or both slots) with the same or very similar names were almost certainly images of each other. These observations aren't true, however, when we compare two different ontologies from different sources. Consider a class named University. In two different ontologies, the class might represent either a university campus or a university as an organization with its departments, faculty, and so on. If we encounter University in two versions of the same ontology, it almost certainly represents exactly the same concept (and because we have a human looking at the results in the end, we can tolerate the "almost" in that sentence).

Comparing ontology versions would be much simpler if we had logs of changes between versions. However, given the decentralized environment of ontology development today, we can't realistically expect such logs to be available. Many ontology development tools provide no logging capability, and ontology libraries are set up to publish ontology versions but not change logs. Representation formats address representation of the ontologies themselves but not changes in them. Furthermore, logs aren't always helpful when several users work on the same ontology. We thus expect users will increasingly need to compare versions without consulting a change log.

PromptDiff heuristic matchers

The PromptDiff algorithm combines an arbitrary number of heuristic matchers, each of which looks for a particular property in the unmatched frames. All the matchers must conform to the *monotonicity principle*: matchers don't retract any matches that have already been established.

The matchers we describe here are fairly simple; our approach's strength lies in their combination. Each looks at a particular part of the ontology structure, such as an is-a hierarchy or slots attached to a class. Being heuristic matchers (hence based on observations that in some cases may not turn out to be true), they could theoretically produce incorrect results. Having examined ontology versions in several large projects, however, we haven't come across such examples and believe the matchers would consistently produce correct results. Furthermore, PromptDiff presents the matching results to a human expert for analysis, highlighting changed frames so that the expert can examine and confirm or reject these matches. These frames usually constitute a small fraction of all frames in an ontology. So, even for very large ontologies, human experts need to examine only a few frames.

Here we describe some of the matchers that we use in the algorithm. In the descriptions below, F_n denotes a frame of any type (class, slot, facet, or instance), C_n denotes a class, and S_n denotes a slot.

The first matcher looks for *frames of the same type with the same name*. In Figure 2, ontology versions V_1 and V_2 have a frame Wine, which in both versions is a class. So the matcher declares that the two frames match. In general, if $F_1 \in V_1$ and $F_2 \in V_2$, and F_1 and F_2 have the same name and type, then F_1 and F_2 match. Frames can be of type class, slot, facet, or instance. In our experiments, this matcher produced on average 97.9 percent of all matches because ontologies usually don't change much from one version to the next.

Another matcher looks for a *single unmatched sibling*. In the example in Figure 2, suppose we matched the classes Wine, Red wine, and White wine from V_1 to their counterparts with the same names in V_2. Then the Wine class in both versions has exactly one unmatched subclass: Blush wine in V_1 and Rosé wine in V_2. In this situation, we conclude that Rosé wine is the image of Blush wine. In general, if $C_1 \in V_1$ and $C_2 \in V_2$, C_1 and C_2 match, and each class has exactly one unmatched subclass ($subC_1$ and $subC_2$, respectively), then $subC_1$ and $subC_2$ match. A similar matcher for multiple unmatched siblings (see the next paragraph) can be distinguished by its set of slots.

We can extend the previous matcher to look for a more complicated situation: *multiple unmatched siblings* exist, but only one pair of siblings has the same set of slots. Suppose that Wine first had only two subclasses, Red and White. Red has a tannin_level slot of type String. In the next version, Wine has three subclasses, and we added "wine" to each subclass name (see Figure 4a). When all of the Wine's subclasses are unmatched, we can still match Red to Red wine because these are the only classes that have the tannin_level slot. In general, if $C_1 \in V_1$ and $C_2 \in V_2$, $C_1$and C_2 match, and $subC_1$ and $subC_2$ are subclasses of C_1 and C_2 respectively, and all the slots of $subC_1$ match all the slots of $subC_2$, and for each of $subC_1$ and $subC_2$, its set of slots is different from the set of slots of all of its siblings, then $subC_1$ and $subC_2$ match.

The next matcher looks for *siblings with the same suffixes or prefixes*. Taking the second situation further, if we remove "wine" from the class name for Wine subclasses (see Figure 4b), all names for these subclasses change. However, if we see they've all changed in the same way—the same suffix has been removed—we can create the corresponding matches anyway. In general, if $C_1 \in V_1$ and $C_2 \in V_2$, C_1 and C_2 match, and all C_1 subclass names match all C_2 subclass names except for a constant suffix or prefix, then the subclasses match.

An *unmatched superclass* provides information for another matcher. Suppose we first used the plural form and called the class at the

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

top of the hierarchy Wines but used singular for all the subclasses (see Figure 4c), then corrected the mistake. (In a separate matcher, we can look for class names that change from plural to singular or vice versa.) All subclasses of the two unmatched classes—Wines and Wine—will then match, and we can conclude that the two unmatched classes match as well. In general, if all subclasses of $C_1 \in V_1$ match subclasses of $C_2 \in V_2$, then C_1 and C_2 match.

Another matcher looks for a *single unmatched slot*. In the example in Figure 2, suppose we matched the class Wine from the first version to its counterpart in the second version. Each class has a single slot that's so far unmatched: maker and produced_by, respectively. Not only is each slot the only unmatched slot attached to its respective class, but the slot's range restriction is also the same: the class Winery. We can therefore match maker and produced_by. In general, if $C_1 \in V_1$ and $C_2 \in V_2$, C_1 and C_2 match, and each class has exactly one unmatched slot, S_1 and S_2 respectively, and S_1 and S_2 have the same facets, then S_1 and S_2 match.

If a knowledge model allows definition of inverse relationships, we can take advantage of such relationships and look for *unmatched inverse slots*. Suppose we have a slot maker in V_1 (at the Wine class in Figure 2), which has an inverse slot makes at the Winery class (see Figure 4d), and we have a slot produced_by in V_2, which has an inverse slot produces. Once we match maker and produced_by, we can match makes and produces because they are inverses of the slots that match. In general, if $S_1 \in V_1$ and $S_2 \in V_2$, S_1 and S_2 match, $invS_1$ and $invS_2$ are inverse slots for S_1 and S_2 respectively, and $invS_1$ and $invS_2$ are unmatched, then $invS_1$ and $invS_2$ match.

Finally, we look for *split classes*. Suppose an early definition of our wine ontology included only white and red wines and we simply defined all rosé wines as White wine instances. In the next version, we introduced a Blush wine class and moved all corresponding rosé wine instances to this new class. In other words, we split the White wine into two classes: White wine and Blush wine. In general, if $C_0 \in V_1$ and $C_1 \in V_2$ and $C_2 \in V_2$, and for each instance of C_0 its image is an instance of either C_1 or C_2, then C_0 splits into C_1 and C_2. A similar matcher identifies merged classes.

Each matcher considers only frames that haven't yet been matched. So in practice each matcher (except the first one) examines only a small number of frames (only those that don't yet have a match).

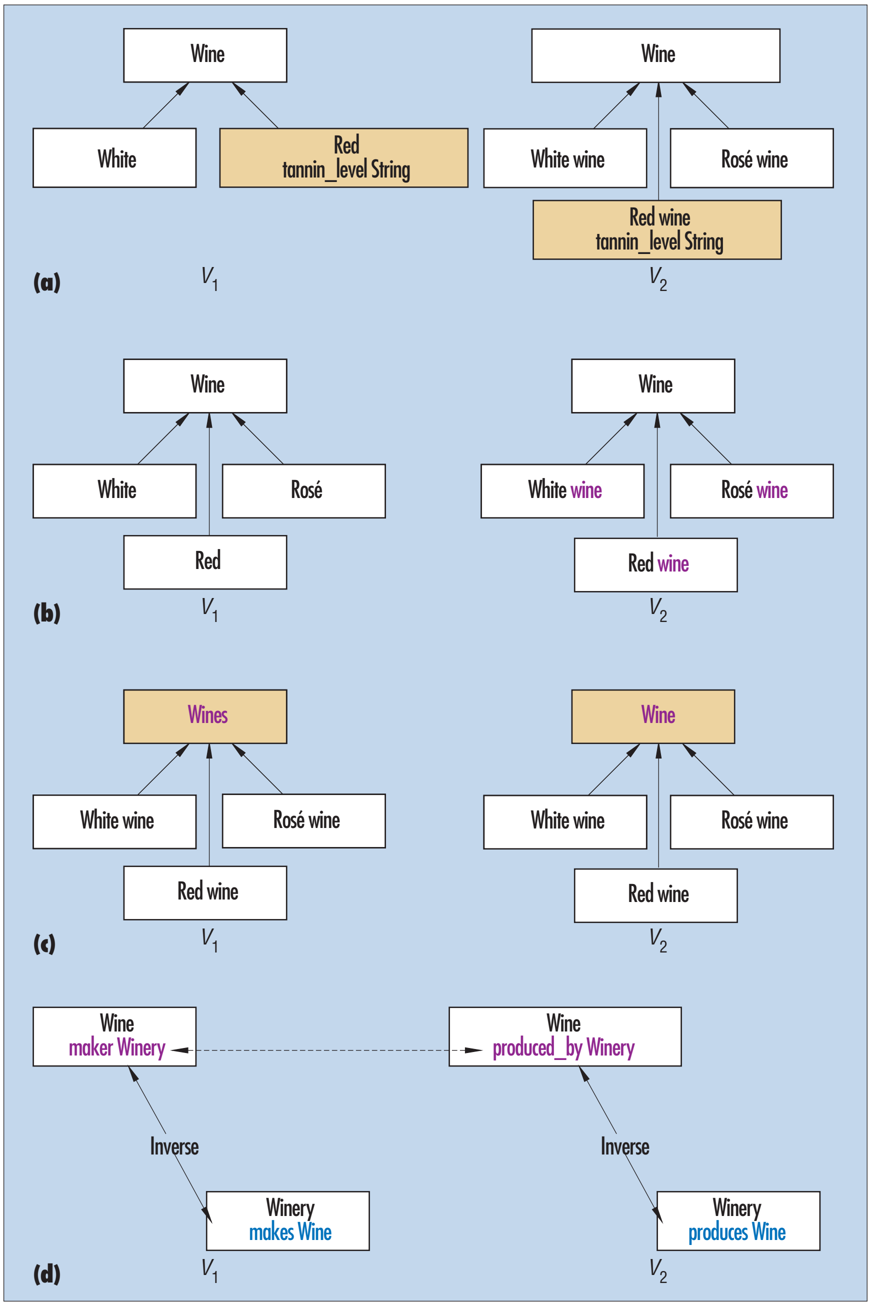


Figure 4. Situations for applying different PromptDiff matchers. (a) The Wine class acquires an additional subclass, and the slots for the class Red in V_1 and the class Red wine in V_2 match. (b) The class names for the Wine subclasses match except for the "wine" suffix. (c) The class names for the Wine and Wines subclasses match; the superclass name has changed. (d) The slots maker and makes are inverse slots in one version; the slots produced_by and produces are inverse slots in another version.

We combine all available heuristic matchers (such as those described earlier, and any others available) in the PromptDiff algorithm, a fixed-point algorithm that produces the complete structural diff for two ontology versions. PromptDiff runs all matchers until they produce no new changes in the table. Because no matcher retracts the results of previous matchers or its own results from previous runs (the monotonicity principle), the algorithm always converges. We show elsewhere that if each matcher's running time is polynomial, the whole algorithm's running time is also polynomial.[4]

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

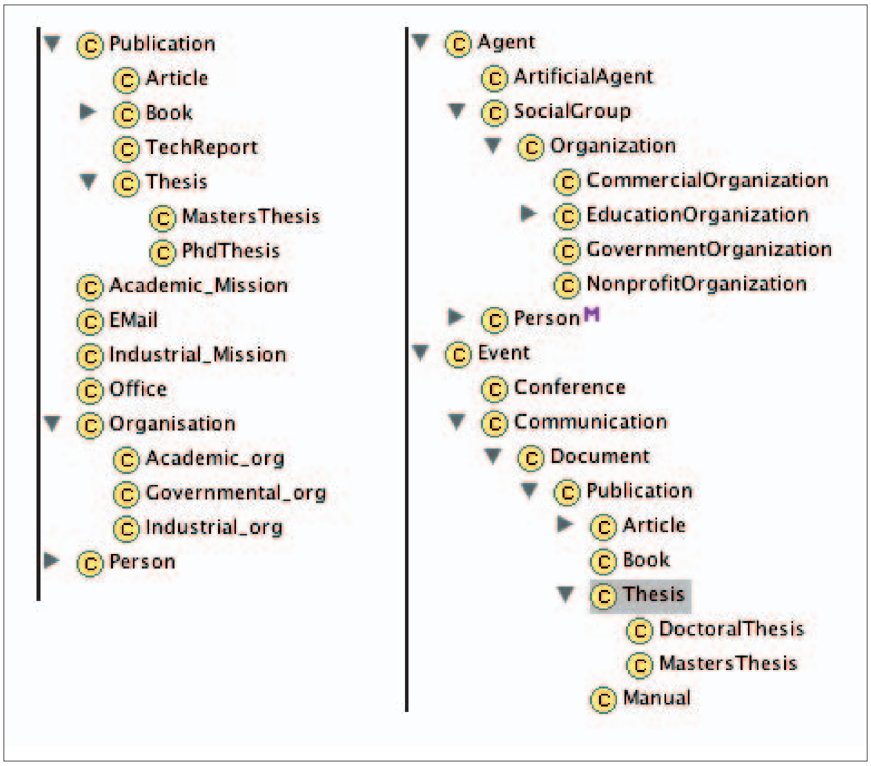


Figure 5. Snapshots of the two source ontologies' class hierarchies, both representing academic organizational structures, developed by DAML groups at (a) Carnegie Mellon University and (b) the University of Maryland.

Evaluation

Empirical evaluation is particularly important for heuristic algorithms because no provable way exists to verify their correctness. We implemented PROMPTDIFF as a Protégé plugin. We then evaluated it using ontology versions in two large projects at our department: EON (www.smi.stanford.edu/projects/eon) and PharmGKB (www.pharmgkb.org). Both rely heavily on ontologies, use Protégé for ontology development, and keep records of different ontology versions. We compared consecutive ontology versions and versions that were farther apart. For each pair, we created the structural diff manually (given that the ontologies contained between 300 and 1,900 concepts, the process was onerous) and compared this manually generated result with the one PROMPTDIFF produced.

We've presented the complete evaluation results elsewhere[4] but summarize them here. On average, 97.9 percent of the frames in each version remained unchanged. To evaluate our algorithm's accuracy, we considered the frames that had changed (the remaining 2.1 percent)—exactly those frames that a user would need to look at. On average, PROMPTDIFF identified 96 percent of the matches between those frames (this measure resembles *recall* in information retrieval terms), and 93 percent of the matches that PROMPTDIFF identified were correct (*precision*). More important, all discrepancies between manual and automatic results were confined to the frames for which PROMPTDIFF did not find any matches. In other words, when PROMPTDIFF did find a match for a frame, it was always correct. Sometimes the algorithm failed to find a match when a human expert could find one. A human expert can determine that two frames are similar even if a rule applied in a specific case isn't sufficiently general to apply in all cases. Two such examples are a significant overlap in a class name (such as Finding versus Physical_Finding) or a similarity in the slot range (such as two slots at matching frames that have the same range). The matchers, in contrast, must use sufficiently general rules that apply to any ontology.

We can interpret these numbers from the user's point of view. In one experiment, V_1 had 1,886 frames, 83 of which changed (in fact, names of 67 of those were accidentally replaced with system-generated names). The PROMPTDIFF result contained 19 unmatched frames; given that we can trust the matches PROMPTDIFF generated, we need to examine only these 19 unmatched frames instead of examining all 1,886 in V_1, a significantly simpler task (it turned out that 14 of those frames had no matches). Hence, even for very large ontologies, users need to examine manually only a small fraction of frames—those for which PROMPTDIFF found no matches. And PROMPTDIFF conveniently shows these frames first in the list of results.

We used 10 matchers in our experiments and, on average, executed each matcher 2.3 times in each experiment. Each matcher produced a new result at least once.

We've also experimented with executing matches in a different order. We found no cases where the final set of matches differed depending on the order in which the matchers executed. Different matchers might identify a particular match each time, but the final set of matches itself remained unchanged. As we extend the set of matchers, we'll likely find cases where the execution order does make a difference.

Versioning and other ontology management tasks

As we previously mentioned, PROMPTDIFF is only one element in our multiple-ontology management infrastructure. Much synergy exists between PROMPTDIFF and ontology-merging tools—we leverage both data and algorithms from one tool in the others. While we designed PROMPTDIFF to compare different versions of the same ontology, IPROMPT helps users find similarities and differences between ontologies from different sources. Many of the heuristic matchers described earlier came originally from IPROMPT. Conversely, PROMPTDIFF provided new heuristics for finding similarities among different ontologies.

Consider, for example, the two ontologies shown in Figure 5. These come from the DAML ontology library and were designed by two different DAML participants. Both ontologies describe the structure of academic organizations.

Consider the matcher for multiple unmatched siblings. If multiple unmatched siblings exist in two different versions, PROMPTDIFF looks at slot information to find matches. In IPROMPT, where we compare two different ontologies, slots aren't likely to be similar, and the exact number of siblings will probably differ as well (compare the two

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.

hierarchies in Figure 5). However, iPROMPT still can use the same heuristic to present two small sets of unmatched siblings to the user, who can then easily identify matches.

For example, suppose we have already matched the following class pairs from Figure 5: Organization and Organization, Governmental_org and GovernmentOrganization, and Academic_org and EducationOrganization. The unmatched siblings are Industrial_org on the CMU side and CommercialOrganization and NonprofitOrganization on the UMD side. Probably no direct match exists between the related classes, but the user might decide that, in fact, Industrial_org and CommercialOrganization are related.

Now consider the matcher for a single unmatched sibling. If the same situation arises during merging, we can suggest to the user, with a high level of confidence, that the classes match. For example, if we match classes Thesis and MastersThesis in Figure 5, classes PhDThesis and DoctoralThesis would be unmatched.

Likewise, iPROMPT can easily reuse many other matchers we've described. If, during merging, iPROMPT encounters a situation similar to that described for unmatched inverse slots, it can also suggest that the user merge the corresponding slots. The matcher that looks for siblings with the same suffixes and prefixes will also work well for merging. The number of siblings could differ slightly because different modelers might represent different divisions. However, iPROMPT can again use the heuristic to indicate a potential match between classes.

Our ontology-versioning and ontology-merging tools differ primarily in the source of matching frame pairs. Whereas PROMPTDIFF collects them by recursively calling different matchers, iPROMPT uses match information provided by users to find new matches. iPROMPT also can use matches produced by ANCHORPROMPT, a graph-based algorithm for comparing different ontologies.[6] Figure 1 shows how the different multiple-ontology management tools interrelate by providing algorithms and data to one another.

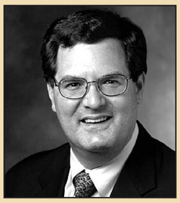
Versioning is just one task in managing multiple ontologies. By looking at these processes in an integrated framework, we can reuse the algorithms and leverage the information we gain in executing one task to perform analysis for another task. For example, in implementing version comparison, we used heuristics that we developed for ontology merging by just lowering the threshold for considering two terms to be similar. Conversely, when studying ontology versions in different projects, we learned new heuristics that we can apply to finding similarities between ontologies for ontology merging and alignment.

When we evaluate interactive merging and mapping tools, we must compare ontologies that different users produced from the same source ontologies. We can use the PROMPTDIFF results to measure similarities between two related ontologies—how many concepts have changed, how many are isomorphic, and so on.

In the future, we plan to explore using PROMPTDIFF results to generate transformation scripts from one version to another (another task in multiple-ontology management). Computer programs can then use these scripts to migrate instance data (as in schema versioning) or to query one version using another version. This task's main challenge will be minimizing the number of lossy transformations, which cause loss of values at intermediate steps. ■

The Authors

Natalya F. Noy is a research scientist at Stanford Medical Informatics at Stanford University, where she works on automatic and semiautomatic tools for managing multiple ontologies. Her research interests include ontology development and evaluation, semantic integration of ontologies, and making ontology development accessible to experts outside computer science domains. She received her PhD in computer science from Northeastern University. Contact her at Stanford Medical Informatics, Stanford Univ., 251 Campus Dr., Stanford, CA 94305; noy@smi.stanford.edu.



Mark A. Musen is a professor of medicine and computer science at Stanford University and heads the Stanford Medical Informatics laboratory. His research focuses on knowledge acquisition for intelligent systems, knowledge system architecture, and medical decision support. He has directed the Protégé project since its inception in 1986, emphasizing the use of explicit ontologies and reusable problem-solving methods to build robust knowledge-based systems. He received his MD from Brown University and his PhD in medical information sciences from Stanford. Contact him at Stanford Medical Informatics, Stanford Univ., 251 Campus Dr., Stanford, CA 94305; musen@smi.stanford.edu.

Acknowledgments

We thank Warren Shen for implementing the Protégé library browser. Parts of this article appeared in the proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002). Protege-2000 is supported by the National Library of Medicine under grant LM007885; the system may be freely downloaded under an open-source license from http://protege.stanford.edu. A contract from the US National Cancer Institute partly supported this research.

References

1. D.L. McGuinness et al., "An Environment for Merging and Testing Large Ontologies," *Principles of Knowledge Representation and Reasoning: Proc. 7th Int'l Conf.* (KR 2000), Morgan Kaufmann, 2000, pp. 483-493.

2. P. Mitra, G. Wiederhold, and M. Kersten, "A Graph-Oriented Model for Articulation of Ontology Interdependencies," *Proc. Conf. Extending Database Technology* (EDBT 2000), LNCS 1777, Springer-Verlag, 2000, pp. 86–100.

3. M. Klein et al., "Ontology Versioning and Change Detection on the Web," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management* (EKAW 02), LNCS 2473, Springer-Verlag, 2002, pp. 197–212.

4. N.F. Noy and M.A. Musen, "PromptDiff: A Fixed-Point Algorithm for Comparing Ontology Versions," *Proc. 18th Nat'l Conf. Artificial Intelligence* (AAAI 2002), AAAI Press, 2002, pp. 744–750.

5. N.F. Noy and M.A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," *Proc. 17th Nat'l Conf. Artificial Intelligence* (AAAI 2000), AAAI Press, 2000, pp. 450–455.

6. N.F. Noy and M.A. Musen, "Anchor-PROMPT: Using Non-Local Context for Semantic Matching," *Proc. Workshop Ontologies and Information Sharing at the 17th Int'l Joint Conf. Artificial Intelligence* (IJCAI 2001), 2001, pp. 63–70; http://CEUR-WS.org/Vol-47.

Authorized licensed use limited to: UNIVERSITY OF ALBERTA. Downloaded on October 1, 2009 at 14:49 from IEEE Xplore. Restrictions apply.