# The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping

Natalya F. Noy and Mark A. Musen

Stanford Medical Informatics, Stanford University,

251 Campus Drive, Stanford, CA 94305, USA

{noy, musen}@smi.stanford.edu

August 5, 2003

**Abstract**

Researchers in the ontology-design field have developed the content for ontologies in many domain areas. This distributed nature of ontology development has led to a large number of ontologies covering overlapping domains. In order for these ontologies to be reused, they first need to be merged or aligned to one another. We developed a suite of tools for managing multiple ontologies. These suite provides users with a uniform framework for comparing, aligning, and merging ontologies, maintaining versions, translating between different formalisms. Two of the tools in the suite support semi-automatic ontology merging: iPrompt is an interactive ontology-merging tool that guides the user through the merging process, presenting him with suggestions for next steps and identifying inconsistencies and potential problems. AnchorPrompt uses a graph structure of ontologies to find correlation between concepts and to provide additional information for iPrompt.

# 1 Managing Multiple Ontologies

Researchers have pursued development of ontologies—explicit formal specifications of domains of discourse—on the premise that ontologies facilitate knowledge sharing and reuse (Musen, 1992; Gruber, 1993). Today, ontology development is moving from academic knowledge-representation projects to the world of e-commerce. Companies use ontologies to share information and to guide customers through their Web sites. Ontologies on the World-Wide Web range from large taxonomies categorizing Web sites (such as Yahoo!) to categorizations of products for sale and their features (such as Amazon.com). In an effort to enable machine-interpretable representation of knowledge on the Web, the WWW Consortium has developed the Resource Description Framework (W3C, 2000), a language for encoding semantic information on Web pages. The WWW consortium is also working on OWL, a more high-level language for semantic annotation on the Web.[1] Such encoding makes it possible for electronic agents searching for information to share the common understanding of the semantics of the data represented on the Web.

Many disciplines now develop standardized ontologies that domain experts can use to share and annotate information in their fields. Medicine, for example, has produced large, standardized, structured vocabularies such as SNOMED (Spackman, 2000) and the semantic network of the Unified Medical Language System (Lindberg et al., 1993). Broad general-purpose ontologies are emerging as well. For example, the United Nations Development Program and Dun & Bradstreet combined their efforts to develop the UNSPSC ontology which provides terminology for products and services (www.unspsc.org).

With this widespread distributed use of ontologies, different parties inevitably develop ontologies

[1] http://www.w3.org/2001/sw/WebOnt/

with overlapping content. For example, both Yahoo! and the DMOZ Open Directory (www.dmoz.org) categorize information available on the Web. The two resulting directories are similar, but also have many differences.

When ontology developers must reconcile disparate ontologies, they need automated or semi-automated help. A domain expert who wants to determine a correlation between two ontologies must find all the concepts in the source ontologies that are similar to one another, determine what the similarities are, and either change the source ontologies to remove the overlaps or record a mapping between the sources for future reference.

Finding correlation between terms in separate ontologies is not the only ontology-management task involving multiple ontologies. *Merging* ontologies is another one. Users may need to integrate two ontologies coming from different sources to create a new ontology that includes information from both sources. As ontology-development becomes a more dynamic and collaborative process, ontology designers need tools for *version management* that would allow them to access different versions, compare them, transform instances.

Researchers in various groups have worked on automated or semi-automated solutions to some of these tasks (see Section 9 for a review). When developing tools to support ontology versioning, ontology merging, ontology alignment, and other tasks in multiple-ontology management in our group, we learned that these tasks are in fact closely related to one another and each of the tools can benefit from being tightly integrated with the other tools in a single framework. More important, the users benefit as well since not only they get better tools, but also they get a uniform user interface, the same interaction paradigm in all the tools, and easy access from one tool to another. Therefore we integrated our tools for multiple-ontology managements into a single

framework. The PROMPT framework includes an ontology-merging tool (iPROMPT), an ontology-alignment tool (ANCHORPROMPT), an ontology-versioning tool (PROMPTDIFF), a tool for factoring out semantically complete sub-ontologies (PROMPTFACTOR), and other tools.

In this paper we describe our tools for ontology merging and alignment in detail, including our formative evaluation of the tools, and give a brief overview of some other tools in the framework.

This paper makes the following contributions:

- We present a uniform view on various tasks in multiple-ontology management.
- We define the PROMPT framework for multiple-ontology management.
- We describe iPROMPT—a component of the PROMPT framework for interactive ontology merging.
- We evaluate the iPROMPT tool in a formative evaluation.
- We describe ANCHORPROMPT—a graph-based algorithm for finding correlations between concepts in different ontologies and a component of the PROMPT framework.
- We evaluate ANCHORPROMPT using ontologies from the DAML ontology library.

## 2 The PROMPT Knowledge Model

Before presenting the PROMPT framework itself, we define the knowledge model underlying PROMPT. In describing the knowledge model, we will use an ontology of academic publications as an example. Our example ontology will include such notions as different types of publications, their authors and titles, dates and places of publication, and so on.

We implemented PROMPT as an extension to the Protégé ontology-editing environment.[2] Thus the PROMPT's knowledge model is greatly affected by the knowledge model of Protégé. However, the PROMPT's knowledge model is more general than that of Protégé and has fewer restrictions: Our goal was to develop tools for multiple-ontology management that would be applicable in a wide variety of formalisms (see Section 8).

The knowledge model of PROMPT (and Protégé) is frame-based: frames are principal building blocks of an ontology. Each frame has a single unique name. We distinguish the following types of frames: classes, slots, and instances.

We define a **class** as a set of entities. **Instances** of a class are elements of this set (the class of an instance is called its **type**). Classes constitute a taxonomic hierarchy with multiple inheritance. If a class $B$ is a **subclass** of a class $A$ (its **superclass**), then every instance of $B$ is also an instance of $A$. For example, a class `Publication` can represent a set of all publications. Its subclass, a class `Book` represents books, all of which are also publications. Each instance can have only one type.[3]

**Slots** are also frames (i.e., slots are first-class objects in our model). When a slot is **attached** to a class (its **domain**), it defines binary relations in which instances of that class can participate in and attributes of the instances. For example, a slot `title` attached to a class `Publication` represents titles of publications. In PROMPT, we assume union semantics for slot domains: if a slot has multiple domains, then instances of all the domain classes have the slot.

A slot can have a **range** which restricts the values a slot can take. A slot range can be another

[2] http://protege.stanford.edu

[3] PROMPT inherited the restriction of a single type for a frame from the knowledge model of Protégé. In fact, this restriction is not essential and all the algorithms can be adapted easily to a knowledge model that allows multiple types for an instance.

class (e.g., a range of a slot **author** is a class `Person`), in which case a slot defines a binary relation between an instance of a class and the slot value (i.e., between an instance of `Publication` and an instance of `Person`). A slot range can also be a primitive datatype (e.g., a range of a slot `title` is a `String`). We define the following primitive datatypes: `String`, `Integer`, `Float`, and `Boolean`. Slot values must belong to the defined range of the slot: if the range is a primitive datatype, slot values must have that datatype; if the range is a class, slot values they must be instances of that class. In the case of multiple range definitions, we assume union semantics: the value of a slot must be an instance of any of the classes in the slot range.

The number of values a slot can have for each instance is limited by the slot's **cardinality**. Each slot has a minimum cardinality that defines the minimum number of values a slot must have and a maximum cardinality which specifies the maximum number of values for a slot. If a maximum cardinality is not defined, the slot can have any number of values.

Slot attachment is inherited from a superclass to its subclasses: a slot attached to a class is also attached to its subclasses. When we attach a slot to a class, its range and cardinality constraints are by default the same as for the frame representing the slot. However, we can further restrict the values locally. For example, suppose a slot `publishedIn`, representing a place where a publication was published, has a range `Publication` (which has such subclasses as `Journal`, `ConferenceProceedings`, and so on). When we attach the slot `publishedIn` to the class `JournalArticle`, we can restrict its range to the class `Journal` (the subclass of the global range `Publication`). Similarly, we can limit cardinality of a slot locally. Local range and cardinality restrictions are inherited to subclasses of a class. They can be further restricted in subclasses.

Note that since slots are first-class objects, their names are in the same namespace as all frame

names and must be unique (and different from names of classes).

Note that we neither require that the sets of classes, instances, and slots to be disjoint, not preclude this feature. In fact, in the Protégé implementation of PROMPT they are not: classes are themselves instances of other classes (their **metaclasses**), and slots are instances of classes (their **metaslots**).

The knowledge model of the Open Knowledge Base Connectivity (OKBC) protocol (Chaudhri et al., 1998) is the closest to the one we described. In fact, the Protégé ontology-editing environment has an OKBC knowledge model. In OKBC, slot restrictions, such as the range of values, minimum and maximum cardinality and others, are described as **facets**. A facet is a ternary relation between a class, a slot, and a value. For example, a `minimum-cardinality` facet linking a class `Publication`, a slot `author` and a value 1 means that each instance of the class `Publication` must have at least one value for the slot `author`. A `value-type` facet linking the class `Publication`, the slot `author`, and the class `Author` means that instances of the class `Publication` must have instances of the class `Author` in their `author` slot. That is, the `value-type` facet defines a local range of the slot. In addition to slot's range and cardinality restrictions, there are facets for defining default values of slots, minimum and maximum values for numeric slots, local documentation of a slot at class, and other properties. Even though for the sake of readability, we mostly discuss only range and cardinality constraints in this paper, our actual implementation of PROMPT treats all facets uniformly (this uniformity also includes user-defined facets, which Protégé allows).

## 3 The PROMPT Framework

As we have already mentioned, we implemented the PROMPT suite of tools for multiple-ontology management as an extension to the Protégé ontology-editing environment. The open architecture

of Protégé allows developers to extend it easily with plugins for specific tasks. We implemented PROMPT as a set of such plugins.

The PROMPT suite includes tools for many of the tasks for multiple-ontology management: interactive ontology merging (Noy and Musen, 2000), graph-based mapping (Noy and Musen, 2001), factoring out semantically independent sub-ontologies, ontology versioning (Noy and Musen, 2002), and ontology-library maintenance. It is through development of these tools that we came to realize that many of these directions are indeed related and started integrating the approaches into a common framework. The tools in the PROMPT suite share user-interface components, internal data structures, some of the algorithms, logging facilities, and so on. Figure 1 shows some of our our existing tools and how they benefit from one another.

iPROMPT is an interactive ontology-merging tool. iPROMPT leads users through the ontology-merging process, suggesting what should be merged, identifying inconsistencies and potential problems and suggesting strategies to resolve them. iPROMPT uses the structure of concepts in an ontology and relations among them as well as the information it gets from user's actions. For example, if iPROMPT's analysis identified that two classes from different ontologies may be similar and then the user merged some of their respective subclasses, iPROMPT will be even more certain that those classes are similar. We discuss iPROMPT in more detail and present results of our evaluation of iPROMPT in Section 4.

iPROMPT uses only *local context* in its decision-making: When analyzing similarities between concepts, iPROMPT looks only at the slots that are directly attached to these concepts and at the concepts that these slots reference through range restrictions. Suppose we view an ontology as a graph where we represent classes as nodes in the graph and slots as edges in the graph con-

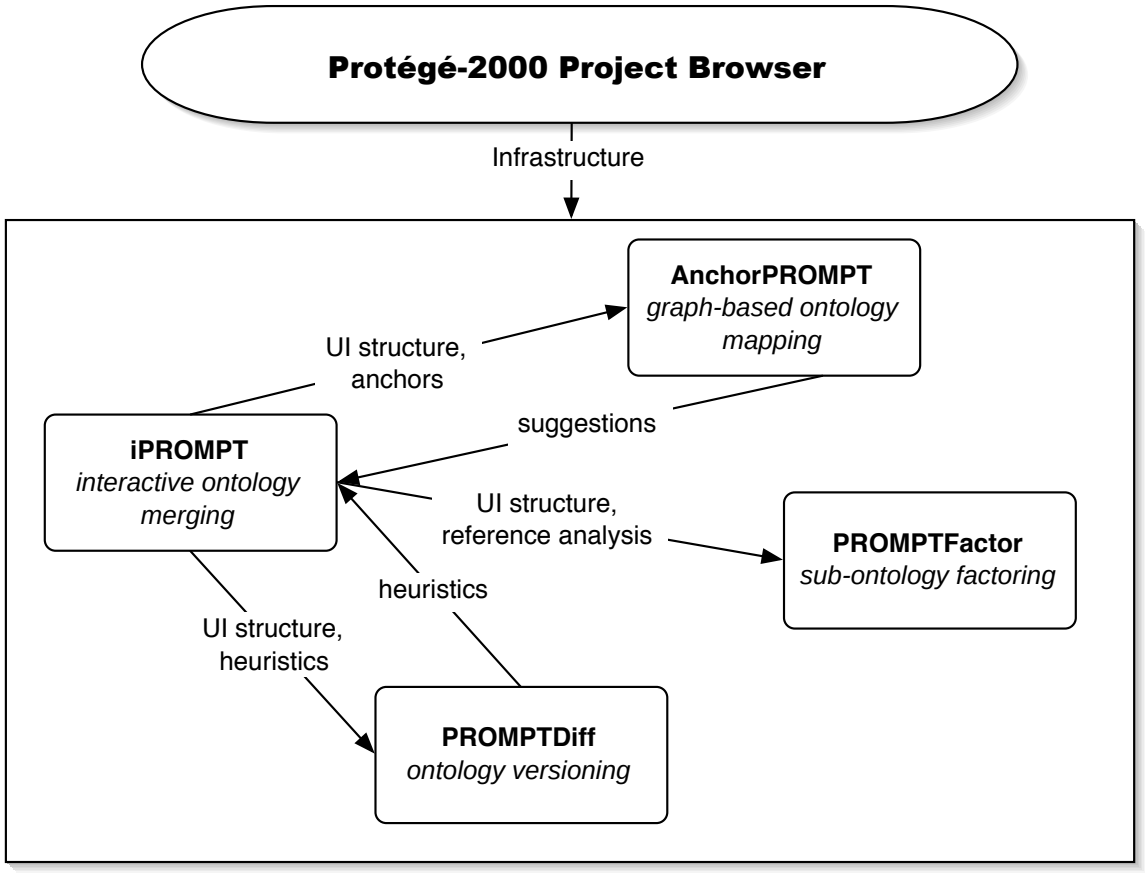



Figure 1: The PROMPT infrastructure and interactions between the tools. The PROMPT suite includes iPROMPT, an interactive ontology-merging tool, ANCHORPROMPT, a graph-based tool for finding similarities between ontologies, PROMPTDIFF, a tool for finding a diff between two versions of the same ontology, and PROMPTFACTOR, a tool for extracting a part of an ontology. These tools share user-interface components, heuristics, data structures, and provide information for one another. The Protégé Project Browser provides overall capabilities for managing multiple ontologies.

necting the nodes representing their respective domains and ranges (Section 5). Effectively, when comparing two concepts in the source ontologies, iPrompt looks only at a very small portion of the ontology graph around each concept, traversing one or two links. AnchorPrompt—another component in the suite—analyzes a graph representing ontologies on a larger scale, producing additional suggestions. AnchorPrompt compares larger portions of the graph than iPrompt does. It takes as input a set of pairs of matching terms in the source ontologies and produces new pairs of matching terms. AnchorPrompt's results could then be used in iPrompt to present new suggestions to the user.

Merging source ontologies to create a new one is not always what the user needs. Sometimes, the users prefer to keep the source ontologies separate for several reasons: compatibility with data and tools using the source ontologies, maintenance of relations between ontologies when the source ontologies evolve, and others. In that case, all that the user may want to do is find and save correspondences between different ontologies. We can use AnchorPrompt in this setting as well to identify pairs of similar concepts that the users must align. We describe AnchorPrompt in Section 5.

PromptDiff is a tool for comparing ontology versions (described in Section 6). We observed that many of the heuristics we used in iPrompt, would be very useful in finding what has changed from one version of ontology to another. These heuristics include analysis and comparison of concept names, slots that are attached to concepts, domains and ranges of slots and so on. At the same time, our level of confidence in the analysis could be much higher than in the case of ontology merging: If two concepts that came from versions of the same ontology look the same (e.g., have the same name and type), they probably *are* the same. Whereas if two frames that came from independently developed ontologies look the same, they may or may not be the same. Consider

a class `University` for example. In two different ontologies, this class may represent either a university campus, or a university as an organization, with its departments, faculty, and so on. If we encounter a class `University` in two versions of the same ontology, it is much more likely that it represents exactly the same concept.

PROMPTFACTOR (Section 6) is a tool that enables users to factor out part of their ontology into a new sub-ontology, making sure that the severed links do not introduce concepts in the subontology that are ill-defined (e.g., do not have a type, refer to frames that do not exist in the ontology, etc.) . The user can specify that he wants all the concepts that are required in the definition of his concept of interest in the new ontology. The tool then performs the transitive closure of the superclass relation and all the relations defined by slots.

We will now describe these tools in detail and come back to interactions between the tools in Section 7.

## 4 iPROMPT—An Interactive Ontology-Merging Tool

Suppose a user has two ontologies covering the same domain that were developed by different groups. Figure 2 presents class hierarchies from two such ontologies. These two ontologies were developed independently by two teams in the DAML project.[4] We imported two ontologies from the DAML ontology library (DAML, ????):

1. An ontology for describing individuals, computer-science academic departments, universities, and activities that occur at them developed at the University of Maryland (UMD), and

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

2. An ontology for describing employees in an academic institutions, publications, and relationships among research groups and projects developed at Carnegie Mellon University (CMU).

The iPROMPT algorithm takes as input two ontologies, such as these ontologies from the DAML library, and guides the user in the creation of one merged ontology as output. Figure 3 illustrates the iPROMPT ontology-merging algorithm. First iPROMPT creates an initial list of matches based on lexical similarity of class names (Figure 4).[5] Then the process goes through the following cycle: (1) the user triggers an operation by either selecting one of iPROMPT's suggestions from the list or by using an ontology-editing environment to specify the desired operation directly; and (2) iPROMPT performs the operation, automatically executes additional changes based on the type of the operation, generates a list of suggestions for the user based on the structure of the ontology around the arguments to the last operation, and determines inconsistencies and potential problems that the last operation introduced in the ontology and finds possible solutions for those problems.

There are several research groups already working on methods for determining linguistic similarity among concept names (see Section 9). Therefore, we decided to concentrate on the steps that analyze the *structure* of the ontology. A specific implementation of the iPROMPT algorithm uses whatever measure of linguistic similarity among concept names is appropriate. In our Protégé-based implementation (Section 4.2), we use the Protégé component-based architecture to allow the user to plug in any term-matching algorithm. In iPROMPT, we start with linguistic-similarity matches for the initial comparison, but concentrate on finding clues based on the semantics of the ontology and user's actions.

[5]The current implementation of iPROMPT uses simple lexical-distance measures to find classes with similar names. However, we designed the tool in such a way that other, more sophisticated term-comparison algorithms could be plugged in. Using WordNet, for example, to find synonyms would be a natural extension.

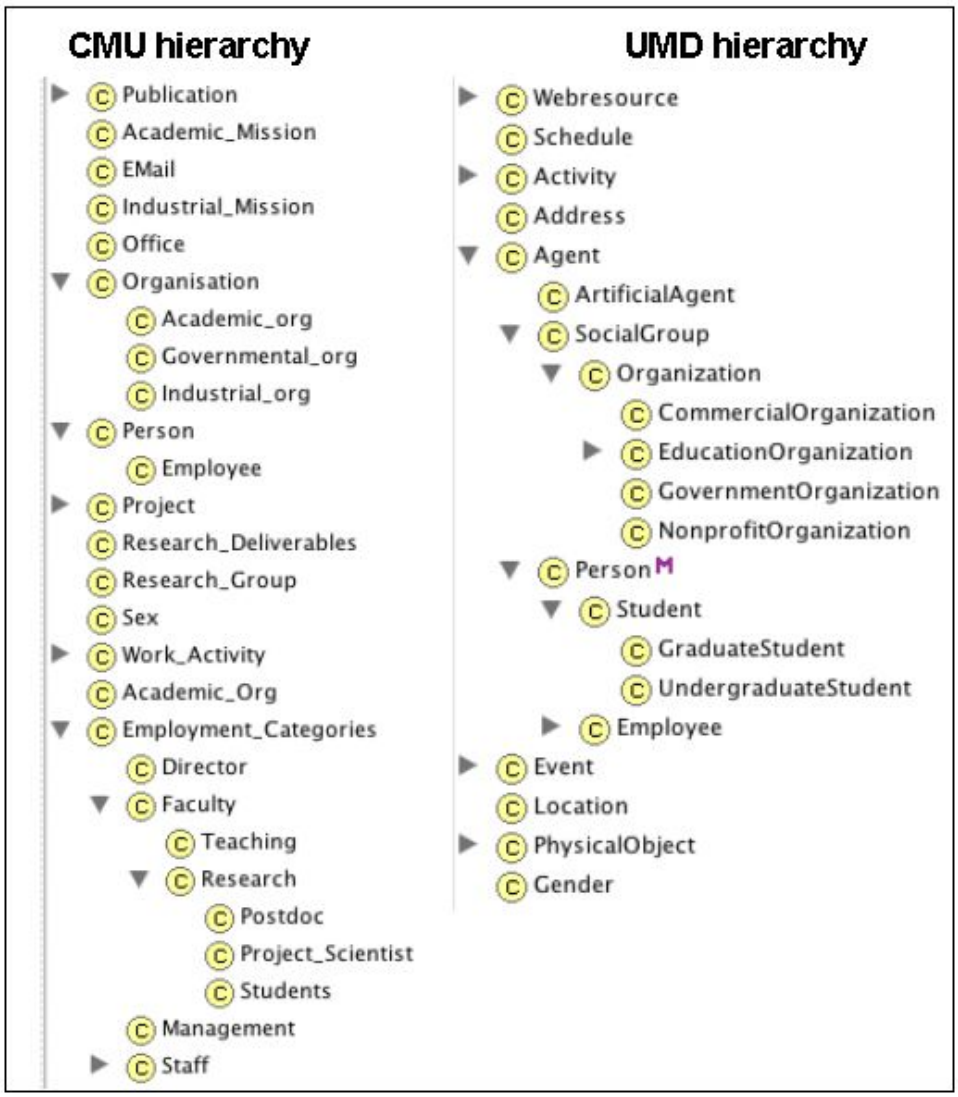



Figure 2: Snapshots of the class hierarchies in the two source ontologies for the experiment. Both ontologies represent structure of academic organizations. They were developed by DAML groups at Carnegie Mellon University (left) and University of Maryland (right). The ontologies are available at http://www.daml.org/ontologies

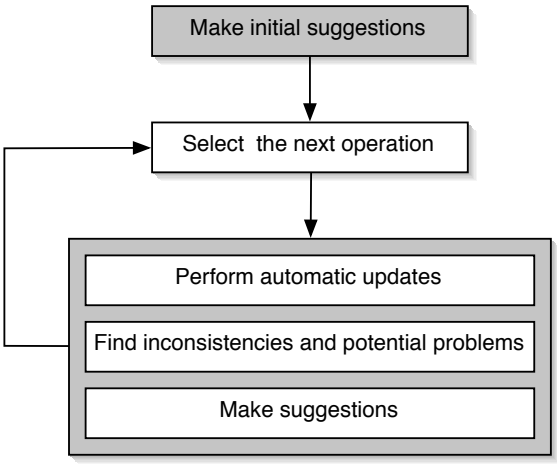



Figure 3: The flow of iPROMPT algorithm.The user selects a new operation. The rest of the actions are performed by iPROMPT.

### 4.1 The iPROMPT Algorithm

The iPROMPT algorithm performs the following **ontology-merging task**: Given two source ontologies, $O_1$ and $O_2$, create a new ontology $O_m$, which we call a **merged ontology**; $O_m$ includes some, but not necessarily all, frames from $O_1$ and $O_2$ with the frames that represent similar concepts merged into one frame. Note that this description of a merging task is necessarily informal since many of the decisions in the merging process rest with the ontology designer and depend on the task for which the designer plans to use the merged ontology. For instance, there may be some parts of the source ontologies that the user will not include in the merged ontology because they cover parts of the domain that are not important for the task at hand. There is also subjectivity in whether or not different frames from the source ontologies represent similar concepts or not. We designed the iPROMPT algorithm to help users make these decisions.

The following is at the heart of our approach: We identify a set of knowledge-base operations for ontology merging or alignment. For each operation in this set, we define (1) changes that iPROMPT

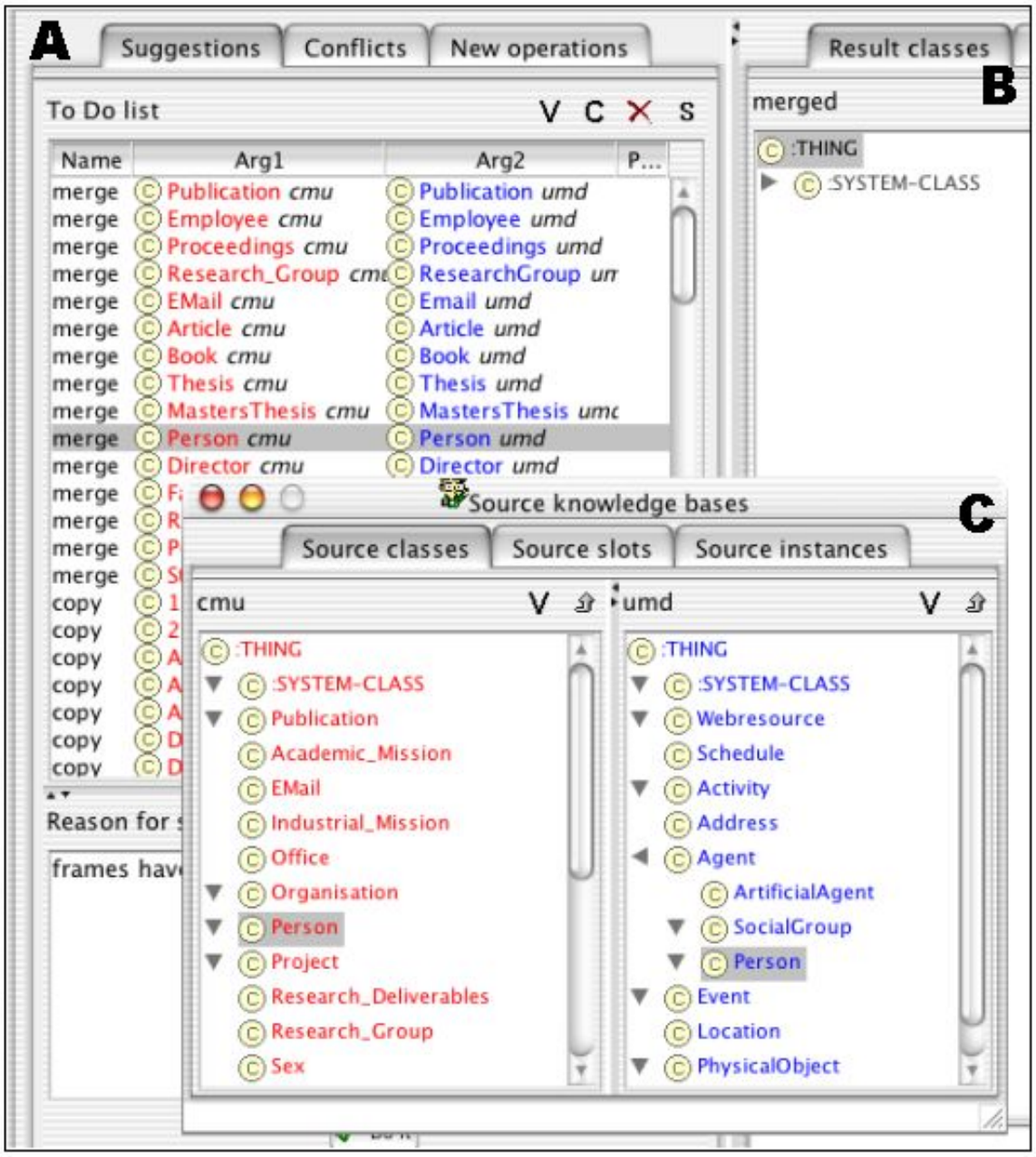



Figure 4: A screenshot of the iPROMPT tool. Part A shows initial iPROMPT's suggestions for the ontologies in Figure 2. Part B displays the emerging merged ontology (which is empty at this starting point). Part C shows the two source ontologies side-by-side. The arguments to the selected operation in the suggestion list (merge two `Person` classes) are selected in the *Sources* window.

performs automatically, (2) new merging suggestions that iPROMPT presents to the user, and (3) inconsistencies and potential problems in the emerging merged ontology that the operation may introduce and that require user's attention. When the user invokes an operation, iPROMPT creates members of these three sets based on the arguments to the specific invocation of the operation.

During the merging process, as iPROMPT (automatically, or with the user's help) copies frames from the source ontologies to the merged ontology, merges frames with each other, and makes changes to the frames in the merged ontology, for each frame in the source ontologies, it keeps track of which frame in the merged ontology it corresponds to. Intuitively, an **image** of a frame $F_s$ from a source ontology is a frame $F_i$ in the merged ontology $O_m$ that represents the same concept in the domain that $F_s$. More formally, we say that a frame $F_i$ in the merged ontology $O_m$ is an image of a frame $F_s$ from a source ontology if:

1. $F_i$ is a result of copying $F_s$ into $O_m$, or
2. $F_i$ is a result of merging $F_s$ with another frame, or
3. $F_i$ is a result of merging an image of $F_s$ with another frame

Note that a frame from a source ontology can have more than one image in the merged ontology.

### 4.1.1 The iPROMPT Operations

The set of ontology-merging operations that we identified includes both the operations that are normally performed during traditional ontology editing of a single ontology and the operations specific to merging and alignment, such as merging classes, merging slots, merging instances, performing a

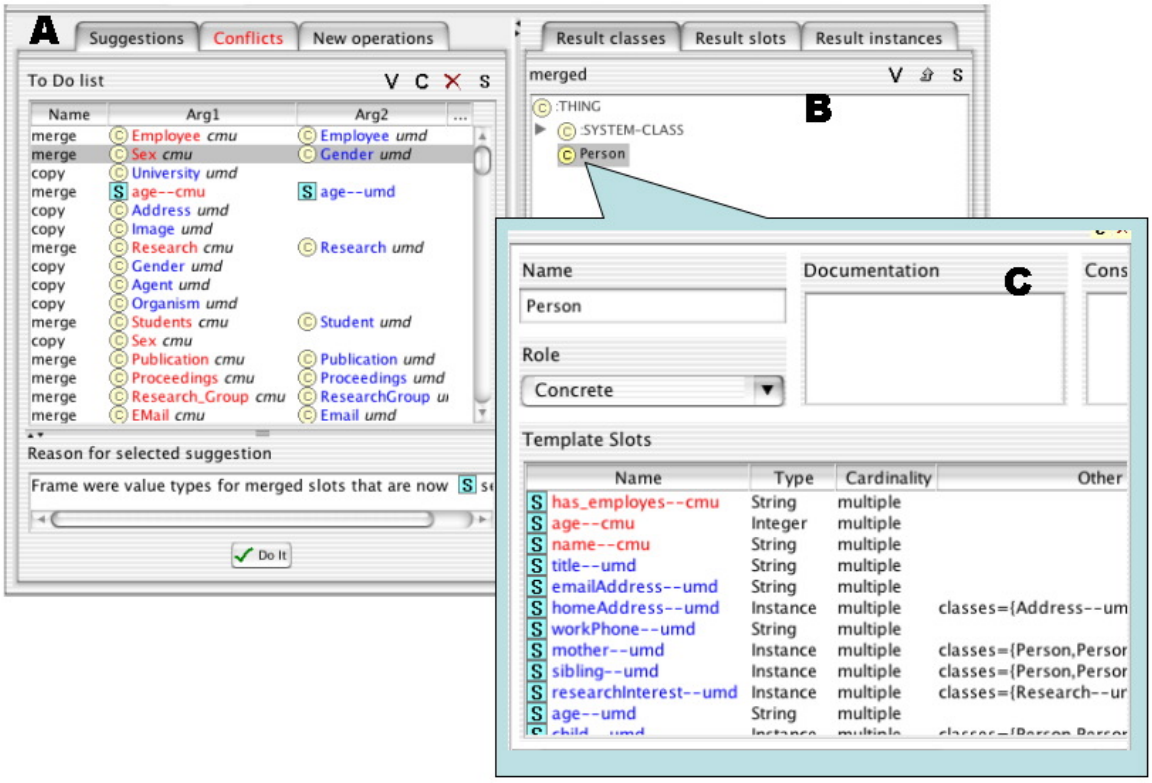



Figure 5: The state of the iPROMPT tool after the user has merged the `Person` classes and the `sex` slots. Part A shows current suggestions. Suggestions at the top of the list refer to frames related to the merged `Person` class. Part B shows the emerging merged ontology, which currently has only one class, `Person`. Part C displays the current definition of the merged `Person` class. This class has slots from both of the source ontologies.

deep or a shallow copy of a class from one ontology to another. In the descriptions of the operations below, $O_m$ is the merged ontology.

**merge classes** Merge two classes. For example, we can follow the system's suggestion and merge two classes `Person` in Figure 4. We describe this operation in detail in Section 4.1.3. Briefly, to merge two classes $A$ and $B$, create a new class $M$ in $O_m$. For each class $C$ that is a subclass or a superclass of $A$ or $B$, if there is an image $C_i$ of $C$ in $O_m$, $C_i$ becomes a subclass or a

superclass of $M$, respectively. For each slot $S$ attached to $A$ or $B$, if there is no image of $S$ in $O_m$, copy $S$ to $O_m$. For each image $S_i$ of $S$, attach $S_i$ to the class $M$. If either $A$ or $B$ was already in $O_m$ prior to the operation, all references to it in $O_m$ become references to $M$ and the original frame is deleted.

**merge slots** Merge two slots. For example, in Figure 5A, the system suggests merging the two slots `age`. This operation merges slots as first-class frames. When merging two slots $S_1$ and $S_2$ create a new slot $S_m$. For each class $C$ in the domain and range of $S_1$ or $S_2$, if there is an image $C_i$ of $C$ in $O_m$, add $C_i$ to the domain or range of $S_m$, respectively. If either $S_1$ or $S_2$ was in $O_m$ prior to the operation, all references to it become references to $S_m$ and the original slot is deleted. Note that as a result of the last step, if there was a class in $O_m$ that had both $S_1$ and $S_2$ attached to it, after the operation it will have only $S_m$ attached to it. For example, after we merge the two `age` slots (Figure 5C), the class `Person` will have the merged slot attached to it. In addition, iPROMPT suggests that the user merges classes in the domain (and range) of $S_m$ that came from different source ontologies. For instance, in one of the source ontologies, a slot `sex` had the class `Gender` as its range and in the other a slot `sex` had a class `Sex` as its range. If we merge the two `sex` slots, iPROMPT will suggest merging classes `Gender` and `Sex`.

**merge instances** Merge two instances. For example, if both source ontologies have instances of the `Proceedings` class describing the proceedings of the AAAI-2002 conference, we can merge these instances when merging the ontologies. iPROMPT does the following when merging two instances $I_1$ and $I_2$ to create a new instance $I_m$: If classes $C_1$ and $C_2$ which are the types of $I_1$ and $I_2$ respectively, have no images in $O_m$, copy them to $O_m$ (see the operation `perform a shallow copy of a class`). If $C_1$ and $C_2$ already have images in $O_m$ and the images are

different frames, merge the images (the user must confirm this operation).[6] Note that as a result, the merged instance $I_m$ will have the same slots, or their images, that $I_1$ and $I_2$ had. For each value $V$ for each slot $S$ attached to $I_1$ or $I_2$, do the following:

- If $V$ is a primitive value (string, number, etc.), add $V$ to the value of the image of $S$ for $I_m$
- If $V$ is a frame and there is an image of $V$, $V_i$, in $O_m$, add $V_i$ to the value of the image of $S$ for $I_m$

For each slot $S$ of $I_m$ that has values that are images of frames coming from different sources, iPROMPT suggests merging these frames. Note that adding images of all the values at sources can create violations of range and cardinality constraints for the merged instance. This inconsistency is one of the inconsistencies that iPROMPT checks for in the next step of the algorithm—finding inconsistencies and potential problems.

**perform a shallow copy of a class** Copy a class from a source ontology to another. For example, after dealing with all the common publication types from the two example ontologies (Figure 4), we can copy the `Manual` class which exists only in the UMD ontology. More precisely, when copying a class $C$, create a new class $C_i$ in $O_m$. For each slot $S$ directly attached to $C$, if there is no image of $S$ in $O_m$, copy $S$ to $O_m$. Then attach images of all the slots of $C$ to $C_i$.

**perform a deep copy of a class** Copy a class from one ontology to another copying all the parents of a class up to the root of the hierarchy. For example, we can copy the `Address` class

[6] This operation is necessary because in the PROMPT knowledge model each frame can have only one type. If we allowed multiple types, the type of the merged instance would be a union of the images of the types of the source instances. iPROMPT would then suggest merging the classes in this set of types that came from different source ontologies

and its slots from the UMD ontology to the merged ontology since the CMU ontology does not contain an `Addess` class. More precisely, to perform a deep copy of a class $C$, perform its shallow copy, and then perform a deep copy for each of its superclasses.

### 4.1.2 Inconsistencies and Potential Problems in the Merged Ontology

While performing the operations that we described in the previous section, iPROMPT checks for the following inconsistencies and potential problems in the merged ontology as a result of these operations. Note that not all of these situations constitute a semantic inconsistency: some of the items in the list below indicate only a potential problem in the merged ontology that requires the user's attention.

**name conflicts** Recall that frame names in an ontology are unique (Section 2). A name conflict occurs when there is more than one frame with the same name in the same ontology. For example, we can copy the class `Location` from the UMD ontology into the merged ontology. If we then copy a slot `Location` that may exist in the CMU ontology, there will be a name conflict.[7] To resolve this priblem, iPROMPT suggests renaming one of the offending frames.

**dangling references** When we bring in frames from one ontology to another, we may not bring all the frames that the moved frames refer to. As a result, a frame will refer to another frame that does not yet exist in the merged ontology. For example, the definition for the merged class `Person` contains a slot `homeAddress` which has the class `Address` as its range (Figure 5C). However, if we have not yet copied the `Address` class, the reference to it will be a dangling one. To resolve thisproblem, iPROMPT suggests copying the dangling frame into

[7] We assume that all frames are in the same namespace

the merged ontology.

**redundancy in the class hierarchy** Links in a class hierarchy may be redundant if there is more than one path from a class to a superclass. For example, consider the source hierarchies in Figure 4. Suppose we merge the two `Publication` classes and the two `Proceedings` classes. We then copy the `Book` class. If we restore all the superclass links, the `Proceedings` class in the merged ontology will have two superclasses, `Publication` (from the CMU ontology) and `Book` (from the UMD ontology). `Book` is also a subclass of `Publication`. Thus there are two paths from `Proceedings` to `Publication`. Note that sometimes the user may actually want such redundancy and therefore may ignore the iPROMPT's suggestion. However, our experience shows that the nature of the merging process leads to the creation of unintended redundancy in the class hierarchy is created very often. Therefore, iPROMPT points this redundancy out to the user, suggests removing one of the offending parents, and leaves it up to the user to decide whether or not to do it.

**slot values violating slot-value restrictions** When we merge instances, the values for slots of the merged instance may violate slot-value restrictions (Section 4.1.1). In particular, range and cardinality constraints can be easily violated when we copy slot values for instances in the slot ontology to instances in the merged ontology.

### 4.1.3 Example: The `merge-classes` Operation

We will now present one of the merging operations, `merge-classes` in detail, listing the changes that iPROMPT makes automatically, new suggestions it makes, and inconsistencies and problems it identifies. Figure 5C shows the result of performing the `merge-classes` operation for two `Person` classes for ontologies in Figure 2.

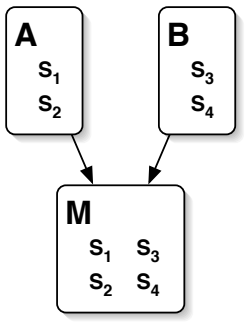



Figure 6: Merging classes $A$ and $B$ to create a class $M$

Suppose the user is merging two ontologies and he performs a `merge-classes` operation for two classes $A$ and $B$ to create a new class $M$ (Figure 6). To execute `merge-classes` operation, iPROMPT then performs the following actions:

- Create a new class $M$. If classes $A$ and $B$ had the same names, assign that name to $M$. Otherwise, ask the user which name to choose (unless he has designated one ontology as preferred).

- For each superclass $C$ of $A$ or $B$ that has an image $C_i$ in $O_m$, make $C_i$ a superclass of $M$ (thus restoring the original relation). Do the same for subclasses. For example, if we have had previously copied the `Agent` class from the UMD ontology (Figure 2) into the merged ontology, the merged `Person` class would become a subclass of this copied `Agent` class. When later we merge the two `Employee` classes (Figure 2), the merged `Employee` class will become a subclass of the merged `Person` class.

- For each slot $S$ that was attached to $A$ or $B$ in the original ontologies, if there is no image of $S$ in $O_m$, copy $S$ to $O_m$. Then, for each slot $S$ that was attached to $A$ or $B$, attach its image to $M$

- If any of the slots $S$ attached to $A$ or $B$ had local cardinality restrictions, copy those restrictions to the attachment of the image of $S$ ($S_i$) to $M$. Similarly, if any of the slots $S$ attached to $A$ or $B$ had a local range restriction $C$ and $C$ has an image $C_i$ in $O_m$, add $C_i$ as a local range restriction for the slot $S_i$ at the class $M$. Therefore, the new `Person` has all the slots that the source classes had. The slots that came from the UMD ontology have the `"umd"` suffix and the slots that came from the CMU ontology have the `"cmu"` suffix (Figure 5C).

- For each pair of slots for $M$ that have linguistically similar names, suggest that the slots are merged. Later, if the user chooses to merge the slots, suggest that the classes restricting the values of these slots, are merged as well. For example, originally the `Person` classes in the source ontologies each had a slot `sex` attached. This slot had the class `Gender` as its range in one ontology, and the class `Sex` in the other. After we merge these two slots, iPROMPT suggests merging the `Sex` and `Gender` classes (see the top suggestion in Figure 5).

- For each pair of superclasses and subclasses of $M$ that have linguistically similar names, suggest that they are merged: These classes have similar names and, in addition, they were both superclasses (or subclasses) for $A$ and $B$, which the user declared to be similar.

- Check for redundancy in the parent hierarchy for $M$: If there is more than one path to any parent of $M$ (other than the root of the hierarchy), suggest that one of $M$s parents is removed.

Note, that iPROMPT bases most of the decisions in the preceding list on the semantics of the concepts and their position in the ontology and not syntax.

When we describe the tool based on the algorithm in the next section, we outline a few other actions that can be taken after each operation, such as rearranging the current list of suggestions to maintain the user's focus.

### 4.2 The iPROMPT Tool

We have described the iPROMPT tool and a sample interaction with it in detail elsewhere (Noy and Musen, 2000). Figure 5 shows the state of iPROMPT after the user has merged the two `Person` classes and the `sex` slots attached to it. Figure 5A shows the suggestions that iPROMPT generated; Figure 5B shows the emerging merged hierarchy, which currently has only one class; Figure 5C shows the current definition of the `Person` class. Here are a few additional features of the iPROMPT tool.

*Setting the preferred ontology.* It often happens, that the source ontologies are not equally important or stable, and that the user would like to resolve all the conflicts in favor of one of the source ontologies. We allow the user to designate one of the ontologies as *preferred*. When there is a conflict between values, instead of presenting the conflict to the user for resolution, the system resolves the conflict automatically, using the value of the preferred ontology.

*Maintaining the user's focus.* Suppose a user is merging two large ontologies and is currently working in one content area of the ontology. We believe that the system's suggestions that the user sees first should relate to the frames in the same area of the ontology in which the user is working and should not force him to change focus to a different part of the ontology. iPROMPT maintains the user's focus by rearranging its lists of suggestions and problems and presenting first the items that include frames related to the arguments of the latest operations.

*Providing feedback to the user.* For each of its suggestions, iPROMPT presents a series of explanations, starting with why it suggested the operation in the first place. For example, iPROMPT may indicate that frames have identical or similar names, frames are both superclasses of classes that have been merged, value types for slots that have been merged (Figure 5) and so on. If iPROMPT later changes the placement of the operation in the suggestions list, it augments the explanation

with the information on why it moved the operation.

*Logging and reapplying the operations.* The process of ontology merging or alignment is not a one-time exercise. After the user has merged or aligned ontologies and perhaps has even developed an application based on the result, the source ontologies may change. This scenario is particularly likely for distributed ontologies developed by independent users. Ideally, reapplication of the merging or alignment process to the changed ontologies should be almost automatic. iPROMPT logs knowledge-level ontology-merging and editing operations. If the original ontologies change, the user only needs to make adjustments to the operations in the log that explicitly refer to the changed frames and the system can then reapply the operations automatically to merge the original ontologies again.

## 4.3 Evaluation of iPROMPT

We evaluated the quality of the suggestions that the tool provides by asking several users to merge two source ontologies using iPROMPT. We recorded their steps, which suggestions they followed, which suggestions they did not follow, and what the resulting ontology looked like.

### 4.3.1 Source ontologies

We used the two ontologies in Figure 2 to evaluate the performance of the iPROMPT merging tool. These two ontologies constituted a good target for the merging experiment because, on the one hand, they covered similar subject domains (research organizations and projects, publications, etc.) and, on the other hand, their developers worked completely independently from one another and therefore there was no intentional correlation among terms in the ontologies. In addition, the domain is easy to understand.

Figure 2 presents snapshots of the two hierarchies. Note that many of the concepts in the two ontologies are similar, but they are represented by different terms: `Industrial_org` versus `CommercialOrganization`, `Governmental_org` versus `GovernmentOrganization`, `Student` versus `Students`, `Organisation` versus `Organization`. The structure of the hierarchy is also different: In the CMU hierarchy, for example, `Students`, `Faculty`, and `Management` are subclasses of the class `Employment_Categories`, whereas in the UMD hierarchy these types of classes are subclasses of `Person`. Even though the CMU hierarchy has a class `Person`, its only subclass is `Employee`.

Given these differences in the sources, the merged ontologies produced by different users will inevitably be different: There are many design decisions that could go either way. For example, will the `Organization` class in the merged ontology be at the top level, as it is in the CMU hierarchy, or will it be a subclass of `SocialGroup`, as it is in the UMD hierarchy? Are the classes `Employment_Categories` from the CMU ontology and `Employee` from the UMD ontology essentially the same classes? The easiest way to answer these questions is to have the designers of the two original ontologies get together and merge them. However, in practice this scenario is unrealistic. Therefore, if our task requires that we merge the two ontologies producing one uniform ontology, the user performing the merge operation will have to make these decisions; different users may make different decisions.

#### 4.3.2 Experiment setup

We asked users to use iPROMPT to merge the two ontologies that we have just described. All the users were previously familiar with ontology design in general and using Protégé to design ontologies in particular. The users had not worked with PROMPT before. None of the users had addressed the problem of merging ontologies prior to the experiment. We believe that this type of

users is a typical user for iPROMPT and similar tools: someone who has a basic understanding of ontology design, has perhaps designed some simple ontologies and now faces the task of merging ontologies.

There were four participants in the experiment—all of them students at the Stanford Medical Informatics who answered our call for participation. Each participant received a package containing:

- the iPROMPT software
- documentation of the tool
- a detailed tutorial
- a tutorial example
- materials for the evaluation

The users performed the evaluation at their convenience on their own computers. We asked each participant to install the tool, to read the tutorial and to follow the examples in the tutorial using the tutorial ontologies. We suggested that they may then look through the documentation to get additional insights. After that, the participants were asked to perform the evaluation itself by merging the two source ontologies. After they were done, they sent back the resulting ontology and the log files for our analysis.

In order to minimize differences in the results, we arbitrarily set up the CMU ontology to be the preferred one. That is, if the users were merging two classes with different names, the name from the CMU ontology was used.

### 4.3.3 Quality of iPROMPT's suggestions

To evaluate the quality of iPROMPT's suggestions, we evaluated precision and recall measures: **Precision** is the fraction of the tool's suggestions that the users decided to follow. **Recall** is the fraction of the operations that the users performed that were suggested by the tool. In our experiments, the average precision was 96.9% and the average recall was 88.6%. The precision was in fact a lot higher than we expected. There are several possible explanations for this remarkable result. First, the users were not experts in ontology merging and did not have any particular task in mind when merging the ontologies. As a result, they found it easier simply to follow the tool's suggestions, as long as they seemed reasonable, rather than to explore the ontologies deeper and to come up with their own operations. Second, there was a significant overlap in the content and structure of the source ontologies, which made automatic generation of correct suggestions easier.

Some of the lower recall figures (it was 49% for `merge` operations in one of the experiments) result from questionable choices that the users made. For example, one user merged `Publication` and `DocumentRepresentation` classes, `EMail` and `ElectronicDocument`, `ProjectScientist` and `Research_Assistant`; slots `publisher` and `publishDate`. iPROMPT also did not identify such pairs of related classes as `Academic_org` and `EducationOrganization` or `Industrial_org` and `CommercialOrganization`.

Even though we did not formally evaluate usability, the fact that all the users were able to use the tool and completely merge the source ontologies after a brief handout tutorial, indicates that the tool is fairly easy to use. Even though we told the participants that while they were still going through the tutorial, they could ask questions about the tool and about the process, only one of them ended up asking a question.

While these experiments demonstrated that iPROMPT can identify a large fraction of related frames, it still uses only very limited part of the graph structure representing an ontology. Given two similar classes, iPROMPT consider classes and slots that are *directly* related to the classes in question. In other words, it uses only the *local context* of the concepts under consideration. In the next section, we describe ANCHORPROMPT, a tool for finding matching concepts in the source ontologies that analyzes larger portions of the graph structure. It uses a set of heuristics to analyze *non-local context* of a concept—other concepts that are several links away from the concept of interest.

# 5 ANCHORPROMPT—Using Non-local Context For Semantic Matching

The goal of ANCHORPROMPT is not to provide a complete solution to automated ontology merging but rather to augment iPROMPT by determining additional possible points of similarity between ontologies. ANCHORPROMPT also provides pairs of related terms to help users in ontology mapping and alignment. ANCHORPROMPT takes as input a set of pairs of related terms—**anchors**—from the source ontologies. Either the user identifies the anchors manually or the system generates them automatically. From this set of previously identified anchors, ANCHORPROMPT produces a set of new pairs of semantically close terms.

ANCHORPROMPT views each ontology $O$ as a directed labelled graph $G$. Each class $C$ in $O$ is represented as a node $G$.For each slot $S$ and for each pair of classes $A$ and $B$ in the domain and range of $S$, respectively, there is a directed edge in the graph from $A$ to $B$. Two nodes connected by an edge in a graph are adjacent. There is a path between two nodes of a graph, $A$ and $B$, if, starting at node $A$, it is possible to follow a sequence of adjacent edges to reach node $B$. The length

of the path is the number of edges in the path. The goal of the AnchorPrompt algorithm is to produce automatically a set of semantically related concepts from the source ontologies using a set of anchor matches identified earlier (manually or automatically) as its input.

To do that, AnchorPrompt traverses the paths between the anchors in the corresponding ontologies. A path follows the links between classes defined by the hierarchical relations or by slots and their domains and ranges. AnchorPrompt then compares the terms along these paths to find similar terms. For example, suppose we identify two pairs of anchors: classes $A$ and $B$ and classes $G$ and $H$ (Figure 7). That is, a class $A$ from one ontology is similar to a class $B$ in the other ontology; and a class $G$ from the first ontology is similar to a class $H$ from the second one. Figure 7 shows one path from $A$ to $G$ in the first ontology and one path from $B$ to $H$ in the second ontology. We traverse the two paths in parallel, incrementing the similarity score between each two classes that we reach in the same step. For example, after traversing the paths in Figure 7, we increment the similarity score between the classes $C$ and $D$ and between the classes $E$ and $F$. We repeat the process for all the existing paths that originate and terminate in the anchor points, cumulatively aggregating the similarity score. The central observation behind AnchorPrompt is that if two pairs of terms from the source ontologies are similar and there are paths connecting the terms, then the elements in those paths are often similar as well. Therefore, from a small set of previously identified related terms, AnchorPrompt is able to suggest a large number of terms that are likely to be semantically similar as well.

### 5.1 Example

To illustrate how AnchorPrompt works, we will consider two ontologies for representing clinical trials, their protocols, applications, and results. The first ontology, the Design-a-Trial (DaT) ontol-

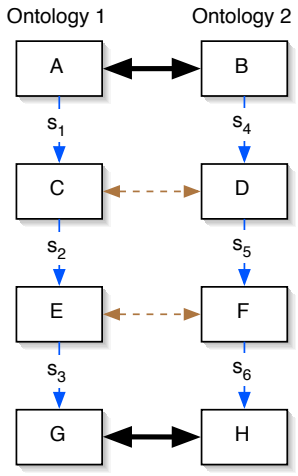



Figure 7: Traversing the paths between anchors. The rectangles represent classes and labeled edges represent slots that relate classes to one another. The left part of the figure represents classes and slots from one ontology; the right part represents classes and slots from the other. Solid arrows connect pairs of anchors; dashed arrows connect pairs of related terms.

ogy (Modgil et al., 2000), underlies a knowledge-based system that helps doctors produce protocols for randomized controlled trials. The second ontology, the randomized clinical-trial (RCT) ontology (Sim, 1997), is used in creating electronic trial banks that store the results of clinical trials and allow researchers to find, appraise, and apply the results. Both ontologies represent clinical trials, but one of them, DaT, concentrates on defining a trial protocol itself, whereas the other, RCT, concentrates on representing the results of the trial. The two ontologies were developed completely independent from each other. Therefore there is no intensional correlation between them. As part of the work on representing clinical guidelines in our laboratory,[8] we needed to merge the two ontologies. Figure 8a shows a part of the graph representing the RCT ontology. For example the edge representing the slot `latest-protocol` in the RCT ontology links the class `TRIAL` (its domain) to the class `PROTOCOL` (its range).

[8] http://smi-web.stanford.edu/projects/eon/

## 5.2 The AnchorPrompt Algorithm

AnchorPrompt takes as input a set of *anchors*—pairs of related terms in the two ontologies. We can use any of the existing approaches to term matching to identify the anchors (Section 9). A user can identify the anchors manually. An automated system can identify them by comparing the names of the terms. For example, we can assume that if the source ontologies cover the same domain, the terms with the same names are likely to represent the same concepts. We can also use a combination of system-determined and user-defined anchors. We can use pairs of related terms that AnchorPrompt has identified in an earlier iteration after the user has validated them. For the example in this section, we will consider the following two pairs of anchors for the two clinical-trial ontologies (the first class in the pair is in the RCT ontology; the second is in the DaT ontology[9]):

```
TRIAL,  Trial
PERSON,  Person
```

Using these two pairs as input, the algorithm must determine pairs of other related terms in the RCT and DaT ontologies. It generates a set of all the paths between `PERSON` and `TRIAL` in the RCT ontology and between `Person` and `Trial` in DaT ontology (Figure 8 shows some of these paths[10]). It considers only the paths that are shorter than a pre-defined parameter length. Now consider a pair of paths in this set that have the same length. For example: Path 1 (in the RCT ontology):

`TRIAL` → `PROTOCOL` → `STUDY-SITE` → `PERSON`

[9]The RCT ontology uses all `UPPER-CASE` letters for class names. The DaT ontology uses `Mixed Case` the class names. Therefore, it is easy to distinguish which class names come from which ontology, and we will sometimes omit the source information.

[10]We have changed the original RCT ontology slightly to simplify this example.

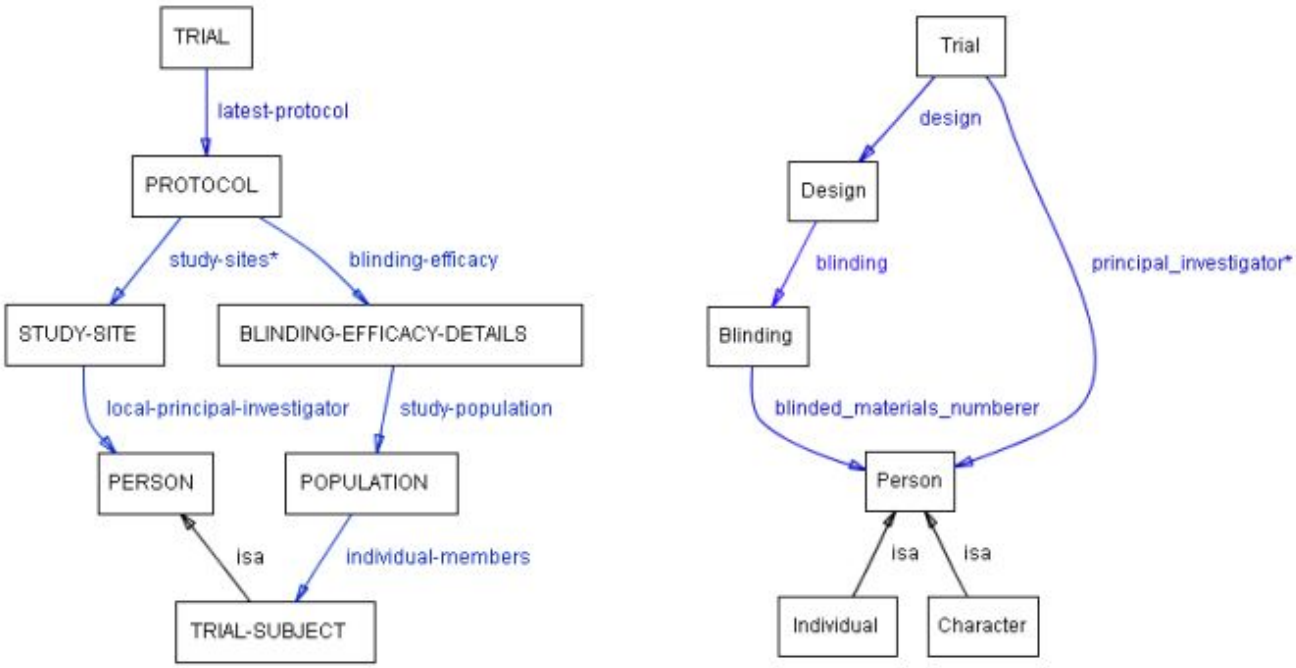



Figure 8: (a) The paths between the classes `TRIAL` and `PERSON` in the RCT ontology; (b) the paths between the classes `Trial` and `Person` in the DaT ontology

Path 2 (in the DaT ontology):

`Trial` → `Design` → `Blinding` → `Person`

As it traverses the two paths, AnchorPrompt increases the **similarity score**—a coefficient that indicates how closely two terms are related—for the pairs of terms in the same positions in the paths. For the two paths in our example, it will increase the similarity score for the following two pairs of terms:

`PROTOCOL, Design`

`STUDY-SITE, Blinding`

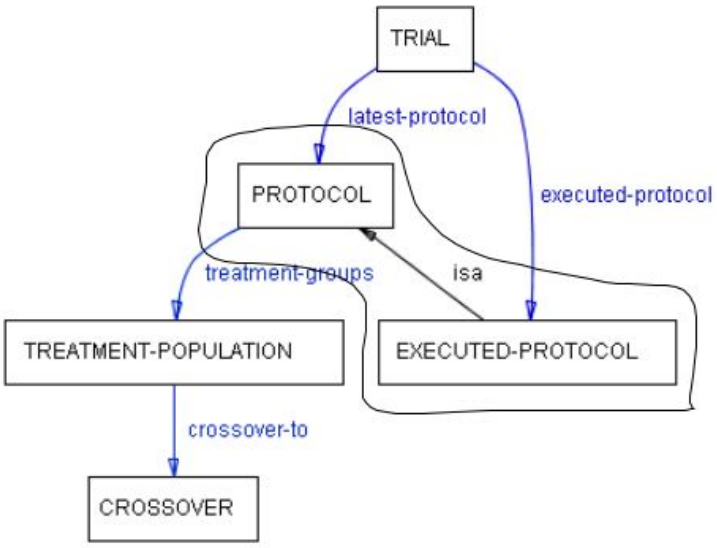



Figure 9: A path from TRIAL to CROSSOVER: The classes EXECUTED-PROTOCOL and PROTOCOL form an equivalence group

AnchorPrompt repeats the process for each pair of paths of the same lengths that have one pair of anchors as their originating points (e.g., `TRIAL` and `Trial`) and another pair of anchors as terminating points (e.g., `PERSON` and `Person`). During this process it increases the similarity scores for the pairs of terms that it encounters. It aggregates the similarity score from all the traversals to generate the final similarity score. Consequently, the terms that often appear in the same positions on the paths going from one pair of anchors to another will get the highest score.

### 5.2.1 Equivalence Groups

In traversing the graph representing the ontology and generating the paths between classes, AnchorPrompt treats the subclasssuperclass links differently from links representing other slots. Consider for example the path from `TRIAL` to `CROSSOVER` in Figure 9.

We could treat the `is-a` link in exactly the same way we treat other slots. However, this approach would disregard the distinct semantics associated with `is-a` links. Instead we can employ the difference in meaning between the `is-a` link and regular slots to improve the algorithm. An `is-a` link connects the terms that are already *similar* (e.g., `PROTOCOL` and `EXECUTED-PROTOCOL`); in fact one describes a subset of the other. Other slots link terms that are arbitrarily related to each other. AnchorPrompt joins the terms linked by the subclasssuperclass relation in **equivalence groups**. In the example in Figure 9, the classes `PROTOCOL` and `EXECUTED-PROTOCOL` constitute an equivalence group. Here is one of the paths from `TRIAL` to `CROSSOVER` in Figure 9 that goes through `EXECUTED-PROTOCOL`. We identify the equivalence group by brackets.

`TRIAL` → `[EXECUTED-PROTOCOL, PROTOCOL]` → `TREATMENT-POPULATION` → `CROSSOVER.`

AnchorPrompt treats an equivalence group as a single node in the path. The set of incoming edges for an equivalence-group node is the union of the sets of incoming edges for each of the group elements. Similarly, the set of outgoing edges for an equivalence-group node is the union of the sets of outgoing edges for each of its elements. The `[EXECUTED-PROTOCOL, PROTOCOL]` equivalence group in Figure 9 has one incoming edge, `executed-protocol`, and one outgoing edge, `treatment-groups`.

The **size** of an equivalence group is the number of class nodes "collapsed" into one. Thus, the equivalence group in the preceding example is of size 2.

### 5.2.2 Similarity Score

We use the following process to compute the similarity score $S(C_1, C_2)$ between two terms $C_1$ and $C_2$ (where $C_1$ is a class from the source ontology $O_1$ and $C_2$ is a class from the source ontology $O_2$).

1. Generate a set of all paths of length less than a parameter $L$ that connect input anchors in $O_1$ and $O_2$.

2. From the set of paths generated in step 1, generate a set of all possible pairs of paths of equal length such that one path in the pair comes from $O_1$ and the other path comes from $O_2$.

3. For each pair of paths in the set generated in step 2 and for each pair of nodes $N_1$ and $N_2$ located in the identical positions in the paths, increment the similarity score between each pair of classes $C_1$ and $C_2$ in $N_1$ and $N_2$ respectively by a constant X. (Recall that $N_1$ and $N_2$ can be either single classes or equivalence groups that include several classes).

Therefore the similarity score $S(C_1, C_2)$ is a *cumulative* score reflecting how often $C_1$ and $C_2$ appear in identical positions along the paths considering all the possible paths between anchors (of length less than $L$).

We change the constant by which we increment the similarity score when the matching nodes along the paths include not single classes but equivalence groups. Suppose we have the following two nodes at the same position on two paths between anchors: $A_1$ and $[B_2, C_2]$, a single class $A_1$ on one side, and an equivalence group with two classes $B_2$ and $C_2$ on the other side. Do we give the same score to both pairs of classes $A_1$, $B_2$ and $A_1$, $C_2$? Is this score the same as the one we would have given the pair $A_1$, $B_2$ had $B_2$ been the only class at the node? Do we give the pairs $A_1$, $B_2$ and $A_1$, $C_2$ any similarity score at all? We analyze the results for different values of these constants in

Section 5.3.

### 5.2.3 Revisiting the Example

We provided AnchorPrompt with the following set of three pairs of anchors from the RCT and DaT ontologies correspondingly:

```
         TRIAL,  Trial
        PERSON,  Person
     CROSSOVER,  Crossover
```

We allowed the paths of length less than or equal to 5 and limited the equivalence-group size to 2.[11] Here are the output results in the order of the descending similarity score.

```
              PROTOCOL,  Design
         TRIAL-SUBJECT,  Person
          INVESTIGATORS,  Person
             POPULATION,  Action_Spec
                 PERSON,  Character
  TREATMENT-POPULATION,  Crossover_arm
```

In fact, all but one of these results represents a pair of concepts that either are similar or one is a specialization (subclass) of the other. The only exception is the pair `POPULATION`, `Action_Spec`. Note that many of these pairings are specific to the domain of clinical trials (e.g., `PROTOCOL`, `Design` and `TRIAL-SUBJECT`, `Person`). The pair `PERSON`, `Character` indeed identifies the correct sense in which `Character` is used in the DaT ontology.

[11] Not all the paths between these anchors appear in Figure 8.

## 5.3 Evaluation of AnchorPrompt

We perform a formative evaluation of AnchorPrompt by testing it on a pair of ontologies that were also developed independently by different groups of researchers. In our experiments, we varied the set of anchor pairs that was the input to the algorithm and various parameters, such as maximum path length, maximum size of equivalence groups, and constants in the similarity score computation. We then analyzed which fraction of the results produced by AnchorPrompt were indeed correct results and which parameter settings produced optimal performance.

We used the same source ontologies as for the evaluation of iPrompt (Figure 2): the ontologies from the DAML ontology library describing research organizations. Figure 10 shows the setup of the experiment. We analyzed the results determining how many of the results were pairs of concepts that were either equivalent or were in a subclass–superclass relationship.

### 5.3.1 Equivalence-group size

If the maximum equivalence-group size is 0 (do not consider subclasssuperclass relationships at all) or 1 (allow equivalence groups of size 1), 87% of the experiments produce empty result sets. If the maximum equivalence-group size is 2, only 12% of the result sets are empty. For the rest of the experiments we fix the maximum equivalence-group size at 2.

### 5.3.2 Similarity score for equivalence-group members

We have conducted two sets of experiments: In the first set all classes in the same position along the path got the same score N and in the second experiment classes that shared their position with

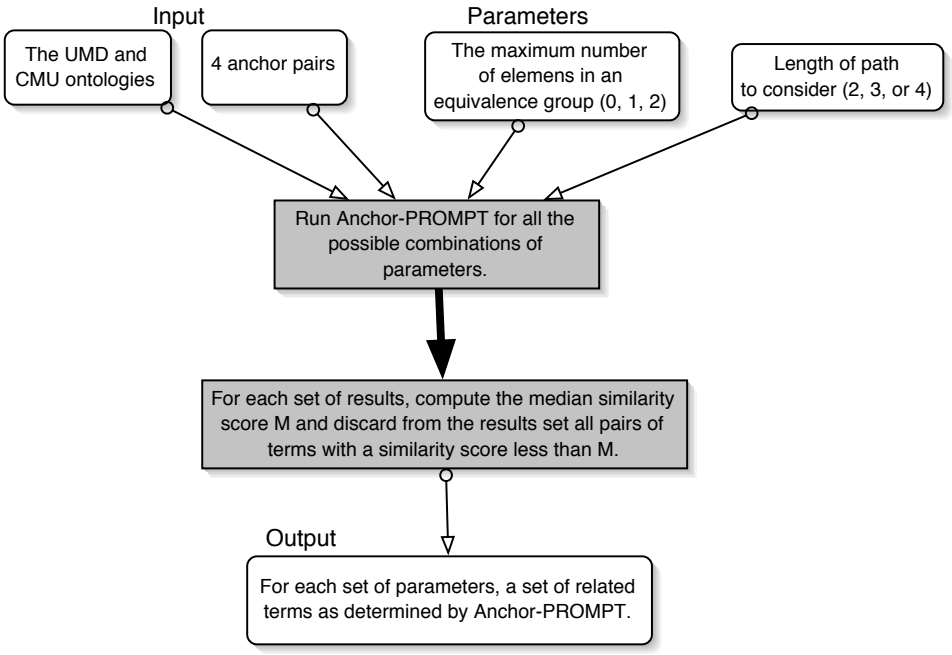



Figure 10: Experimental setup for the evaluation of AnchorPrompt. We ran AnchorPrompt on the same set on input data with varying parameters.

other members of an equivalence group received only 1/3 of the score.[12] Differentiating the score improved the correctness of results by 14%.

For the rest of the experiments we reduced the scores for members of equivalence groups.

### 5.3.3 Number of anchor pairs and maximum length of path

Table 1 presents results for various values for the two remaining parameters: the number of anchor pairs that were input to an experiment and the maximum allowed length of the path. For these experiments (as well as for a set of other experiments with different source ontologies), we received the best result precision (the highest ratio of correct results to all the returned results) with the maximum length of path equal to 2. When we limit the maximum path length to 3, we achieve the average precision of 61%. The precision goes up slightly (to 65%) with maximum path length of 4.

## 5.4 Discussion Of AnchorPrompt

To understand the intuition behind AnchorPrompt consider paths of length one (Figure 11a). Recall that the length of a path is the number of edges in the path. If class $A$ is similar to class $A'$ and class $B$ is similar to class $B'$, it is plausible to assume that the slots connecting them, $s$ and $s'$, are similar as well. In Figure 11b, we introduce an additional class, $C$ and $C'$ correspondingly, on the path. We get the paths of length 2. We continue the analogy by assuming that there is an increased likelihood that $C$ and $C'$ are similar. In addition, slots $s$ and $s'$ and $p$ and $p'$ are similar (AnchorPrompt does not currently record the similarity among slots; we plan to add this feature

[12] In fact, varying the fraction of the score that we assigned to equivalence-group members has not changed the result: The results were identical for equivalence-group scores that were 1/3 or 1/2 of the score for single classes.

| Max path length | Number of anchors | Result precision |
|---|---|---|
| 4 | 4 | 67% |
| 4 | 3 | 67% |
| 4 | 2 | 61% |
| 3 | 4 | 67% |
| 3 | 3 | 61% |
| 3 | 2 | 56% |
| 2 | 4 | 100% |
| 2 | 3 | 100% |
| 2 | 2 | 100% |

Table 1: Result precision with respect to maximum path length and the number of anchors.

in future versions). The algorithm is based on the assumption that developers link the terms in the ontology in a similar manner even if they do not call the terms with the same names. Therefore, very long paths are unlikely to produce accurate results. As the path that we traverse becomes longer, it becomes less likely that they represent the same series of terms and relations. Very short paths, however, consistently produced extremely small (but also extremely precise results sets). For the maximum path length of 2, AnchorPrompt produced result sets that contained only one pair of terms (with a similarity score above the median for that set) but this pair was always a correct one.

Limiting the path length by 1 will produce the results that are equivalent to iPrompt's results (Section 4).

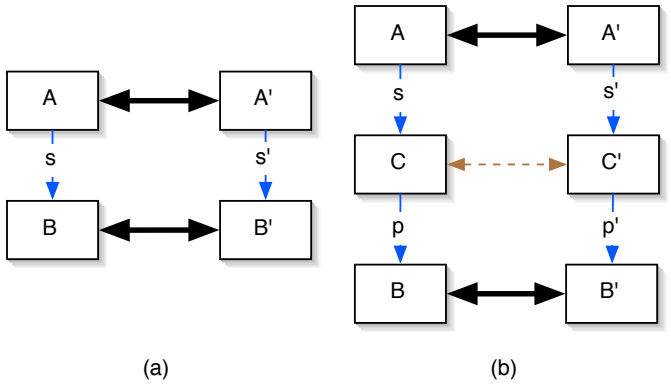



Figure 11: The simple case: the paths of length 1 and 2.

### 5.4.1 Reducing the effect of negative results

The similarity score between concepts is a cumulative similarity score: AnchorPrompt combines the score along all the paths. As a result, we reduce the effect of false matches: Two unrelated terms could certainly appear in identical positions in one pair of paths (and usually do). However, the same two unrelated terms are less likely to appear in identical positions on a different pair of paths. To remove these incidental matches, we determine the median similarity score in each experiment and discard the pairs of terms with a similarity score less than the median. Therefore, AnchorPrompt will discard most of the incidental pairs of terms because they would have appeared only once in the identical positions and would have a low similarity score.

### 5.4.2 Performing ontology mapping

Throughout our discussion we have referred to the process of ontology merging, the process in which we start with two source ontologies and generate a new ontology that includes and reconciles all the information from the two source ontologies. However, the approach that we have presented

can be used directly for creating a mapping between terms in ontologies, as well as in matching database schemas. The result of the AnchorPrompt algorithm is a set of pairs of similar terms ranked by how close to each other the terms are. This result can be used either to trigger merging of the closely related terms or to establish a mapping between the terms.

### 5.4.3 Limitations

AnchorPrompt produced highly promising results with two sets of ontologies that were developed entirely independently from each other. Our approach does not work equally well for all ontologies however. The approach does not work well when the source ontologies are constructed differently. For example, we used AnchorPrompt to find related terms in two ontologies of problem-solving methods: (1) the ontology for the unified problem-solving method (UPML) development language (Fensel et al., 1999) and (2) the ontology for the method-description language (MDL) (Gennari et al., 1998). Both ontologies describe reusable problem-solving methods, however, their designers used different approaches to knowledge modeling. The UPML ontology has a large number of classes with slots attached to and referring to classes at many different levels of hierarchy. The MDL ontology has a lot fewer classes with the hierarchy which is only two levels deep. If we think of the ontologies in terms of a graph, many of the nodes from the UPML ontology were collapsed in a single node in the MDL ontology. As a result, no two pairs of anchors had paths with the same length between them and the output of AnchorPrompt was empty. In general, AnchorPrompt does not work well when one of the source ontologies is a deep one with many inter-linked classes and the other ontology is a shallow one where the hierarchy has only a few levels and most of the slots are associated with the concepts at the top of the hierarchy, and very few notions are reified. If this is the case, the results produced by the algorithm are no different from the results produced

by the approaches that consider only very local context.

# 6 Other Tools In The PROMPT Suite

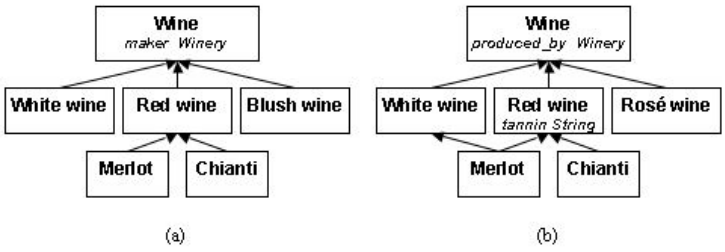
In this paper, we concentrate on describing our tools for finding correlation between ontologies that come from different sources but cover the same or similar domains. We will describe some other tools in the PROMPT suite for managing multiple ontologies only briefly.

## 6.1 Ontology Versioning With PROMPTDIFF

As ontology development becomes a more ubiquitous and collaborative process, the developers face the problem of maintaining versions of ontologies akin to maintaining versions of software code in large software projects. Version comparison to identify the changes between versions (finding a **diff**) is a common operation in such projects. However, version comparison in software code is based on comparing text files—an approach that does not work for comparing ontologies. Two ontologies can be identical but have different text representation. PROMPTDIFF is a heuristics algorithm that produces a **structural diff** between two versions. PROMPTDIFF compares the structure of the two ontology versions and identifies which frames did not change at all, which frames changed only their properties, and which frames changed their names, properties, and other parts of the definition. We describe the PROMPTDIFF algorithm in detail elsewhere (Noy and Musen, 2002). We present only a brief overview of the algorithm here.

Suppose that we are developing an ontology of wines. In the first version (Figure 12a), there is a class `Wine` with three subclasses, `Red wine`, `White wine`, and `Blush wine`. The class `Wine` has

a slot `maker` whose values are instances of class `Winery`. The class `Red wine` has two subclasses, `Chianti` and `Merlot`. Figure 12b shows a later version of the same ontology fragment. Note the changes: we changed the name of the `maker` slot to `produced_by` and the name of the `Blush wine` class to `Rosé wine`; we added a `tannin level` slot to the `Red wine` class; and we discovered that `Merlot` can be white and added another superclass to the *Merlot* class. Figure 12c shows the differences between the two versions in a table produced automatically by PROMPTDIFF. The first two columns are pairs of matching frames from the two ontologies. Informally, given two versions of an ontology $O$, $V_1$ and $V_2$, two frames $F_1$ from$V_1$ and $F_2$ from$V_2$ **match** if $F_1$ became $F_2$. Other columns in the table provide more information about the match. The third column identifies whether or not the frame has been renamed. The last two columns specify how much the frame has changed, if at all.




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

(c)

Figure 12: Two versions of a wine ontology (a and b) and the PROMPTDIFF table showing the difference between the versions

The PROMPTDIFF algorithm consists of two parts: (1) an extensible set of heuristic matchers and (2) a fixed-point algorithm to combine the results of the matchers to produce a structural diff between two versions. Each matcher employs a small number of structural properties of the ontologies to produce matches. The fixed-point step invokes the matchers repeatedly, feeding the results of one matcher into the others, until they produce no more changes in the diff.

Our evaluation has showed that 93% of the matches that PROMPTDIFF produces are correct and that it finds 96% of correct matches.

## 6.2 PROMPTFACTOR—Factoring out Sub-ontologies

PROMPTFACTOR is an example of the PROMPT tool that uses the infrastructure of the framework to provide a useful service to ontology designers. PROMPTFACTOR does not employ any fancy algorithms or provide additional automation. However, it simplifies a lot some of the tasks that an ontology engineer performs.

Suppose we have a large ontology of some domain. For example, we participated in the development of an ontology of human anatomy which included more than 80,000 concepts in one of our projects (Rosse et al., 1998; Noy et al., 2002). One of the design choices was not to break up the ontology into smaller parts and so it was one very large model. Later some of the users wanted to reuse only a part of this ontology, for example, a part describing a human breast (for applications related to breast cancer). They wanted to explore only that relatively small part without having to import the huge ontology of all the human anatomy. Therefore, what they needed was a sub-ontology that included only the concepts relevant to describing a breast. However, this sub-ontology must contain all the necessary definitions, so that it can be used in a stand-alone fashion, without referencing the complete ontology. We call the task of separating such semantically independent sub-ontology from a larger ontology **factoring of sub-ontologies**.

With PROMPTFACTOR, users can identify the term from the larger ontology that they are interested in (`Breast` in our example), and simply press a button to have the tool copy all the terms that are necessary to define the selected term into a separate ontology. PROMPTFACTOR traverses the

ontology starting with the selected term and determines transitive closure of all relations, including *subclass-of* relation. The latter brings in all the parents of the selected term in the hierarchy. The only relation that PROMPTFACTOR treats differently from others is the *superclass-of* relation: it brings in all the subclasses of the selected term, but does not traverse *superclass-of* relation for any other term. It is easy to see that if it did that, it would always bring in the whole ontology: the transitive closure of the *subclass-of* relationship includes the root of the hierarchy. The transitive closure of the *superclass-of* relation of the root includes all the classes.

PROMPTFACTOR also enables users to be more selective in what they want included in the sub-ontology. Instead of asking PROMPTFACTOR to bring in all the related term, they can proceed in selecting themselves the terms that they want to see in the sub-ontology. PROMPTFACTOR then uses the same mechanism for finding inconsistencies as IPROMPT does and presents the list of all the dangling references to the user (Once again, we see the close relationship between different tools in the PROMPT framework).

# 7 Interactions Between The PROMPT Tools

Figure 1 shows some of the interactions among the tools in the PROMPT suite. Each of the tools benefits from some of the elements that we developed for other tools. IPROMPT, as the first tool that we developed, provides overall look-and-feel of the tool, user-interface components, and data structures to other tools. The user-interface elements include presentation of relations between concepts in different sources, the ability to examine concepts from different ontologies and everything related to them directly from the tool, the ability to see and browse the source ontologies side-by-side, the ability to see arguments to any of the relations in their respective places in the

original hierarchies (Figure 4).

Furthermore, iPROMPT can provide anchors for ANCHORPROMPT. Recall that ANCHORPROMPT needs pairs of related terms from the source ontologies to get started. iPROMPT can provide these pairs of terms to ANCHORPROMPT, either by using the information that users supply or by using the results of its own analysis. After ANCHORPROMPT analyzes the graph structure and produces its own set of pairs of related terms, iPROMPT can then present these pairs to the users as additional suggestions.

On the iPROMPT–PROMPTDIFF axis, the two tools exchange heuristics. Most of the heuristics in the initial PROMPTDIFF implementation were the same heuristics we used in iPROMPT: PROMPTDIFF simply lowered the threshold for being certain that two terms are related. For example, if the tools know that two classes are matched, then PROMPTDIFF will assert that their superclasses should be matched as well (assuming they haven't been matched yet). At the same time, iPROMPT will simply suggest to the user that it looks like these superclasses are similar and maybe the user wants to merge them. Later, as we examined more ontology versions in the PROMPTDIFF work, we implemented new heuristics for determining whether two concepts should be matched, and incorporated those heuristics in the iPROMPT analysis as well.

Even though the tools in the PROMPT suite are designed for different tasks in multiple-ontology management, having them in the same framework not only relieves some of the burden on the user since the user-interface paradigm and the interactions are similar in all the tools, but also, improves each of the tools by enabling reuse of both heuristics and code from one tool to another.

# 8 Using The PROMPT Tools For Other Knowledge Models

As we have mentioned in Section 2, we designed the PROMPT tools based on a knowledge-model definition that contained only a small number of primitives and restrictions. Our goal was to make the approach as widely applicable as possible. In fact, the approach's simplicity and its reliance on a simple knowledge model is one of its strength. We believe that the techniques and heuristics we described here can be carried over to managing ontologies defined in many formalisms.

One requirement that we make is that the model describing the domain must be structured. Therefore, PROMPT is directly applicable to frame-based formalisms. The knowledge model of OKBC is representative of frame-based knowledge model and encompasses the features that many frame-based formalisms have. There are several features of the OKBC knowledge model (that are also present in Protégé) that we have not discussed in this paper, mainly for simplicity. However, PROMPT does handle these features within its framework. Metaclasses is one such feature. A **metaclass** is a class whose instances are themselves classes. In fact, every class is an instance of some metaclass. A class `:CLASS` is a class of all classes. Other metaclasses are subclasses of `:CLASS`. Therefore, each class in OKBC has dual identity: it is a class, but it is also an instance. There are also two types of slots: template slots and own slots. **Own slots** associated with a frame describe direct properties of the frame. For example, slots describing properties of individual instances of a class are own slots (e.g., `title` of an instance of `Publication`). **Template slots**, which can be associated only with a class, describe properties of instances of the class. Thus, because of its dual identity, each class has two sets of slots: own slots describing properties of the class itself (in its identity as an instance) and template slots describing properties of its instances. When we referred to slots attached to classes in the paper, we were referring to template slots. In the context of merging *instances*, we use own slots. For many operations however, the PROMPT tools

consider both the own slots and template slots in uniform manner. For example, when iPROMPT determines which other frames a class frame refers to (to update its list of suggestions), it analyzes both domain, range, and other facets of the class's template slots as well as values for its own slots. Similarly, PROMPTDIFF compares values of own slots as well as domain, range, and other facets of template slots to determine the degree of similarity between frames. PROMPTFACTOR looks for domain, range, and other facets of template slots and for values of own slots. Likewise, the tools do not need to distinguish between metaclasses and "regular" classes and can treat them in the same manner. Just as types of individual instances are merged before instances themselves can be merged, if iPROMPT is merging two classes with different metaclasses, it will merge the metaclasses first. Furthermore, as we have mentioned, the PROMPT tools treat all the facets uniformly.[13] Therefore, if users define new facets, the tools will use those facets in the analysis as well.

Frame-based formalisms are not the only formalisms that can be used in the PROMPT framework. Because of the simplicity of the underlying knowledge model, we can also use our approach with ontologies developed in formalisms that are not frame-based. PROMPT can be easily adapted to formalisms based on description logics (DL), where it can use the domain structure that automated DL reasoners infer. Consider, for example, such DL-based languages as the Web Ontology Language (OWL) (Dean et al., 2002) and its predecessors, OIL (Fensel et al., 2000), and DAML+OIL (Hendler and McGuinness, 2000). While OWL is not a frame language itself, many of its primitives translate into a frame formalism quite naturally (Bechhofer et al., 2001). We can treat classes in OWL in the same way as we treat classes in our model. Properties in OWL are similar to slots: they are binary relations between instances of classes or an instance of a class and XML datatype. The notion of domain, range, and a cardinality of a property is similar to the one we use in PROMPT.[14]

[13] ANCHORPROMPT is one exception as it uses only the domain and range of slots in constructing the graph

[14] Note that the semantics of multiple domains and ranges in OWL is different however: OWL (and its predecessors)

It is important to note that in OWL (and other languages based on description logic) a class can have more than one definition. We need to rely on automated reasoners to "bring together" different parts of a class definition. Furthermore, since classes in description logics are often defined by their necessary and sufficient conditions, we also need to rely on a subsumption reasoner to establish a complete class hierarchy. Then, if we have ontologies (or two versions of the same ontology) defined in OWL and an inference engine has produced a class hierarchy and class definitions, we can use this hierarchy and definitions in the PROMPT tools to find similarities and differences between the ontologies. The main feature that the PROMPT heuristics are looking for is whether or not classes are related and whether, for example, values of a particular property (role) are limited to instances of a particular class. Thus, suggestions that the PROMPT tools produce will be relevant for analyzing description-logics ontologies as well. Similarly, we can use PROMPTFACTOR to carve out a part of an OWL ontology and to identify and sever links with the original ontology. In the future, we plan to consider if we can use the inference mechanisms for description logics to assist in the analysis of similarities and differences and in the evaluation of the results of merging and factoring.

The requirement that the model describing a domain must be structured is hard to avoid though. Thus, for example, the tools will not be applicable to a set of axioms in first-order logic. It is worth noting however, that if we consider representation systems that impose some structure on a domain modelled as a set of first-order logic axioms (e.g., CycL (Lenat, 1995), OOFOL (Amir, 1999)), the

assume intersection semantics for domains and ranges of properties. However, many of the same approaches and heuristics that we describe for PROMPT, can apply. Both in IPROMPT and ANCHORPROMPT we base our heuristics on the fact that two classes are related through a range restriction or a subclass definition and (in ANCHORPROMPT) whether or not such a relationship is transitive. The same heuristics will apply whether multiple relationships using the same slot have union or intersection semantics.

PROMPT tools can use this limited structure in the analysis.

# 9 Related Work

In recent years, researchers developed a number of tools to support management of multiple ontologies. These tools include tools for merging ontologies, for defining declarative alignment rules, for finding correspondences between terms automatically. There has also been some theoretical work on defining a general framework for expressing mappings between ontologies. So far, however, these two types of research—tool-oriented and theoretical—come from different research groups. We hope that the two directions will converge in the future. We start our overview of the related work by discussing the theoretical direction. We then describe several tools that were developed as well as related work in the area of database-schema integration.

## 9.1 Theory

Theoretical work in defining correlations between multiple ontologies has centered around developing a formal model for expressing mappings and expressing queries. Bernstein and colleagues (Bernstein et al., 2000) suggest a general way of expressing mappings between models. In their work, a model is any complex structure, such as an XML DTD, UML model, relational schema, semantic network, or an ontology. The authors posit that once we define a mapping between two models, we can easily perform any operation requiring a correlation of the models: merge them, transform instances from one to another, pose queries to one model using another, find a diff between models, and so on. Since a definition of mapping is independent of the way the models themselves are expressed, this structure can be used to map, for example, between a relational schema and an XML

DTD. Mappings are first-class objects that themselves can be manipulated. A complete mapping between two models is a complex structure including expressions that map between elements of the models. A map may be a simple equivalence expression or can include a complex expression or a complex structure on either side. The model is very general and allows for both directional and non-directional mappings. A mapping declaring an equivalence of a result of an SQL query over one model to a result of an SQL query over another model, is non-directional. A mapping which includes an expression for converting instances from one model to instances in another is of course directional. The model also allows one-to-one as well as many-to-one and many-to-many mappings. The authors concentrate on using one-to-one mapping though. The main challenge in this work will be to translate a general model of mappings to practical expressions of mappings for different models that would be useful for tasks of merging, differencing, data transformation and so on.

Madhavan and colleagues (Madhavan et al., 2002) take a similar approach by defining a general approach to mappings, which can then be applied to models of different kinds. They present a well-defined semantics for mappings and define various properties of mappings. These properties include: query answerability (using a mapping to answer queries over one model in terms of another); mapping inference (determining whether two mappings are equivalent), and mapping composition (obtaining a mapping that is a composition of two other mappings). Even though authors present their mapping semantics for a general case that includes ontologies as the models being mapped, they prove that these properties are decidable only in the case of a mapping between two relational-database schemas (it is unclear however if these properties are intractable). The next step will be to apply these properties to more complex models, such as ontologies. It is yet unclear whether such an extension could have a practical implementation.

Calvanese and colleagues (Calvanese et al., 2001) address the problem of integrating local ontologies

into a global ontology. The authors propose using description logics (DLs) to express the ontologies themselves. They argue however, that even the most expressive DLs are not sufficient to express mappings between concepts and propose to use queries (or views) to define mappings. The authors explore two approaches: global-centric and local-centric. In a global-centric approach concepts in the global ontology are expressed as queries over concepts in the local ontologies. In the local-centric approach, concepts in the local ontologies are mapped to queries over the global ontology. The framework that the authors develop, leaves open the question of how to answer queries in such system efficiently. We discuss one of the implemented systems, ICOM, which uses this framework in a specific setting, in Section 9.3.

Whereas the formal models of mappings described above deal with mappings between particular elements of the source models, ontology algebra (Mitra et al., 2001) defines operators for ontologies themselves. Given a set of mapping rules ("articulation rules" in ontology-algebra terms), Mitra and colleagues define intersection, union, and difference between two ontologies. Articulation rules are logical rules that express correspondences between some of the terms in the source ontologies. These rules usually include only the terms relevant to particular application—no effort to map ontologies completely is made. The assumption here is that often we do not need to know the mapping between all the terms in the source ontologies: Only a small portion of both ontologies is relevant to an application that uses them both. Given the articulation rules between two ontologies, their *intersection* is then all the terms and relations that participate in the articulation rules. A *union* of two ontologies is the union of all the concepts and relations from the two sources. A *difference* between two ontologies is the first ontology without the concepts and relations that comprise the intersection. The ONION tool, which we describe in the next section, enables domain experts to define articulation rules, which are used in the algebra definitions.

A natural extension of the PROMPT suite would be to integrate these theoretical models with the implementation. For example, as the users perform interactive ontology merging with iPROMPT, the system can define an explicit mapping between the tools in the background. Other agents could then use this mapping to transfer data from one ontology to another or to query one ontology in terms of the other.

## 9.2 Ontology-Merging And Mapping Tools

We can classify tools that deal with finding correspondences between ontologies into four groups (Figure 13):

- Tools for merging two ontologies to create a new one (e.g., iPROMPT, Chimaera (McGuinness et al., 2000), OntoMerge (Dou et al., 2002))
- Tools for defining a transformation function that transforms one ontology into another (e.g., OntoMorph (Chalupsky, 2000))
- Tools for defining a mapping between concepts in two ontologies by finding pairs of related concepts (e.g., ANCHORPROMPT, GLUE (Doan et al., 2002), OBSERVER (Mena et al., 2000), FCA-Merge (Stumme and Mädche, 2001))
- Tools for defining mapping rules to relate only relevant parts of the source ontologies (e.g., (Mitra et al., 2001))

Another way to categorize the tools is by the type of input on which the tool relies in its analysis and which it requires:

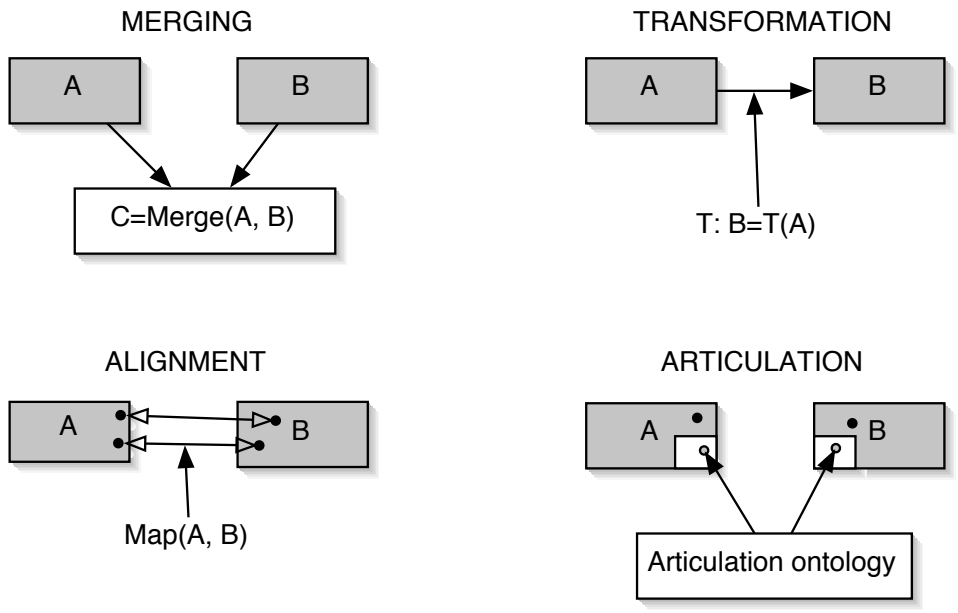



Figure 13: Different types of tools for integrating ontologies.

- class names and natural-language definitions (e.g., tools developed at ISI/UCS for semi-automatic alignment of domain ontologies to a large central ontology (Hovy, 1998))
- class hierarchy (e.g., Chimaera)
- class hierarchy, slots, and facets (e.g., iPrompt, AnchorPrompt, ONION)
- instances of classes (e.g., GLUE and FCA-Merge)
- descriptions of classes (e.g., tools based on description logic, such as OBSERVER)

We will now describe various tools that we mentioned in more detail.

Chimaera (McGuinness et al., 2000) is an interactive merging tool based on the Ontolingua ontology editor (Farquhar et al., 1996). Chimaera allows a user to bring together ontologies developed in different formalisms. The user can request an analysis or guidance from Chimaera at any point during the merging process. The tool will then point him to the places in the ontology where his

attention is required. In its suggestions, Chimaera mostly relies on which ontology the concepts came from and, for classes, on their names. For example, Chimaera will point a user to a class in the merged ontology that has two slots derived from different source ontologies, or that has two subclasses that originated in different ontologies. Chimaera leaves the decision of what to do entirely to the user and does not make any suggestions itself. The only taxonomic relation that Chimaera considers is the subclass–superclass relation. Chimaera is the tool closest to iPROMPT. However, since it uses only a class hierarchy in its analysis, it misses many of the correspondences that iPROMPT finds. These correspondences include suggestions to merge slots with similar names that are attached to merged classes, to merge domains of slots that were merged, and so on. In addition, since Chimaera relies on ontolingua user interface for class editing, the interface is quite cumbersome and hard to use for many developers.

In OntoMerge (Dou et al., 2002), a merged ontology is a union of two source ontologies and a set of *bridging axioms*. The first step in the OntoMerge merging process is to translate both ontologies to a common syntactic representation in a language developed by the authors. Then an ontology engineer defines bridging axioms, which are axioms containing terms from both ontologies. Then instance-translation process looks as following: all instances in the source ontologies are considered to be in the merged ontology. Then an inference engine draws conclusions based on the statements in source ontologies and bridging axioms, thus creating new data in the merged ontology. OntoMerge provides tools for instance-data translation to the merged ontology. It can probably be used in conjunction with tools such as iPROMPT or ANCHORPROMPT to help ontology engineers write bridging axioms.

Ontomorph (Chalupsky, 2000) defines a set of transformation operators that can be applied to an ontology. A human expert then uses the initial list of matches and the source ontologies to define

a set of operators that need to be applied to the source ontologies in order to resolve differences between them, and Ontomorph applies the operators. Therefore, aggregate operations can be performed in a single step. However, a human expert receives no guidance except for the initial list of matches.

GLUE (Doan et al., 2002) is a system that employs machine-learning techniques to find mappings. GLUE uses multiple learners exploiting information in concept instances and taxonomic structure of ontologies. GLUE uses a probabilistic model to combine results of different learners. The learners that GLUE uses currently relies on ontologies having instances and they work much better if many slot values have text in them rather than references to other instances.

The OBSERVER (Mena et al., 2000) system uses description logic to answer queries using multiple ontologies and information on mappings between them. First, users define a set of inter-ontology relations. The system helps the users with this task by looking for synonyms in the source ontologies. Having defined the mappings, users can pose queries in description-logic terms using their own ontology. OBSERVER then employs the mapping information to pose queries to source ontologies. OBSERVER relies heavily on the fact that descriptions in the ontologies and the queries are *intensional* and therefore works best in a DL setting.

FCA-Merge (Stumme and Mädche, 2001) is a method for comparing ontologies that have a set of shared instances or a shared set of documents annotated with concepts from source ontologies. Based on this information, FCA-Merge uses mathematical techniques from Formal Concept Analysis (Ganter and Wille, 1999) to produce a lattice of concepts which relates concepts from the source ontologies. The algorithm suggest equivalence and subclass–superclass relations. An ontology engineer can then analyze the result and use it as a guidance for creating a merged ontology.

However, the assumption that two ontologies to be merged share a set of instances or have a set of documents where each document is annotated with terms from both sources is a very strong one and in practice such a situation may occur only rarely. As an alternative, authors suggest the use of natural-language–processing techniques to annotate a set of documents with concepts from the two ontologies.

The ONION system (Mitra et al., 2001) is based on the ontology algebra that we described in Section 9.1. Therefore, it provides tools for defining articulation rules between ontologies. The articulation rules usually take into consideration only small relevant parts of the source ontologies. ONION uses both lexical and graph-based techniques to suggest articulations. The method to find lexical similarity between concept names uses dictionaries and semantic-indexing techniques based on co-occurrence of words in a text corpus. For graph-based matching, some of the techniques that ONION uses are similar to the ones we described for iPROMPT: look for nodes that have many attributes in common, look for classes with common parents, and so on.

Hovy (Hovy, 1998) describes a set of heuristics that researchers at ISI/USC used for semi-automatic alignment of domain ontologies to a large central ontology. Their techniques are based mainly on linguistic analysis of concept names and natural-language definitions of concepts. (There is a very limited use of taxonomic relationships as well). First, the matcher uses natural-language–processing techniques to split composite-word names (a common occurrence in concept names). It then compares substrings of different lengths to find concept names that are similar to each other. The second consideration are the words used in the natural-language definitions of concepts. The matcher stems the words in the definitions, removes stop words, but keeps the duplicates. It then compares the number and the ratio of shared words in the definitions to find definitions that are similar. An experimentally determined formula for combining these measures of similarity yields

potential matchers that the user needs to examine and approve.

This diversity in the types of tools makes it difficult to compare them directly. In fact, when a developer needs to choose a tool to use in management of multiple ontologies, which tool is the most appropriate will depend on the task at hand (among the tasks in Figure 13) and what type of input the user has. For example, if the ontologies to be merged share a set of instances, then FCA-Merge may work best. If ontologies have instances but don't share them and many slot values contain text, GLUE may be the best choice. If only parts of ontologies need to be mapped, ONION could be the tool of choice. If ontologies have very limited structure but concepts have detailed natural-language definitions (in the same language), the ISI/USC tools may provide the best answers. If no instances are available and ontologies contain many relations between concepts, iPROMPT or ANCHORPROMPT may work best.

## 9.3 Related Work In Schema Integration

This discussion of related work would be incomplete without a brief overview of the work in schema integration. Database schemas are similar to ontologies: Both are structural representations of knowledge. The difference often lies in *scale* rather than substance: Database schemas in practice use much fewer modeling primitives than ontologies and are often smaller than ontologies. Just as with ontologies, researchers are often faced with the problem of finding correlation between different schemas. However, the more common approach in the database-schema research is to develop *mediators* rather than find point-to-point correlations. Users pose their queries in terms of the hand-made mediator schema; queries are then translated into the terms of each of the schemas.

However, in recent years there has been a significant body of work on matching database schemas

directly and on doing this matching automatically. Schema matching for schema integration is similar to ontology mapping: In both cases, we need to find related concepts in two knowledge structures. At the same time, most of the schema-integration techniques consider either relational-database schemas or schemas containing trees of concepts, and perhaps, concept attributes. Rahm and Bernstein (Rahm and Bernstein, 2001) survey schema-integration approaches. They define a taxonomy of these approaches and we can see many similarities to the classification of ontology-merging approaches that we discussed in Section 9.2. Some schema-matching techniques rely on instance data (e.g., LSD (Doan et al., 2001)) and some use only the schema itself (e.g., ARTEMIS (Castano et al., 2001), Cupid (Madhavan et al., 2001)). There are matchers that use linguistic information, such as synonyms, hypernyms, and homonyms (e.g., ARTEMIS, DIKE (Palopoli et al., 1999)), others use machine-learning techniques (e.g., LSD). Most schema machers produce one-to-one matchers, just as most ontology matchers do. A lot of power comes from hybrid approaches that integrate different techniques. There are also interactive tools for schema matching that allow users themselves to specify matching rules (e.g., Clio (Miller et al., 2000)). So, the two fields clearly can benefit from each other. The main difference between schema-matching and ontology-matching approaches (PROMPT in particular) remains the number of knowledge-modeling primitives on which the analysis relies. Schema-matching approaches do not usually consider domains and ranges of slots and complex properties. They also usually look only for matches between concepts, whereas in ontology matching finding correspondences between properties is just as important.

ICOM (Franconi and Ng, 2000) is one example of a project that spans both the schema-integration and ontology-integration domains. It provides a conceptual model and a reasoning mechanism for schema integration, although it does not actually help designers in finding the mappings themselves. ICOM is a tool that allows users to define extended Entity-Relationship diagrams and to specify

declaratively inter-schema constraints, effectively defining mappings between schema (Calvanese et al., 1998a). In ICOM, users can define entities and relationships as views over other entities and relationships using $\mathcal{DLR}$ as a view language, which is a description-logics language over unary and $n$-ary relationships (Calvanese et al., 1998b). The same formalism is used to define both intraschema and interschema relationships. ICOM then treats the resulting system, including the different schema and the interschema relationships, as a single DL knowledge base to perform inference.

# 10 Concluding Remarks

In this paper, we described the PROMPT framework for managing multiple ontologies. The framework includes tools for interactive ontology merging, ontology alignment, versioning, and other tasks. We showed that our approaches and algorithms are effective through formative evaluation of the tools. We also showed that we benefit from integrating the tools for carrying out different tasks in multiple-ontology together.

We intentionally defined the knowledge model underlying PROMPT to have as few elements and as few restrictions as possible. Since many existing knowledge-representation systems are compatible with this knowledge model, the solutions we developed in PROMPTcan be applied to a variety of knowledge-representation systems.

For future work, we plan to leverage further the advantages of the uniform view at different ontology-management processes by introducing explicit declarative specifications of correlations between source ontologies, such as the ones described in Section 9.1. So far, the theoretical work and implementation of practical tools in this area have come from different research groups. However,

just as we benefited from bringing together different types of tools, of them dealing with tasks in multiple-ontology management, we believe we can benefit by including elements of the work that has been theoretical so far as foundation for our tools. As users use our interactive tools, we can be instantiating explicit specification of mappings between ontologies in the background. We can then use these explicit mappings to go from merging to alignment, or to avoid having to repeat the manual part of the process if source ontologies change.

Our vision is that ultimately users will be able to develop new ontologies, reuse existing ones, find mappings between them, merge ontologies, maintain different versions, separate ontologies and then bring them back together, all in one integrated framework, seamlessly switching between the tasks. We believe that this paper has described significant strides toward implementing that vision.

## Acknowledgments

We are very grateful to Monica Crubézy, Ray Fergerson, and Samson Tu for their many suggestions, their help with the tool, and for their comments on earlier versions of the paper. We are also grateful to anonymous reviewers for their insightful feedback. We would also like to thank everyone who participated in the evaluation experiments. Parts of this paper where published at the Seventeenth National Conference on Artificial Intelligence (AAAI-2000) and at the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001). This work was supported in part by the grants 5T16 LM0733 and 892154 from the National Library of Medicine, by a grant from Spawar, by a grant from FastTrack Systems, Inc. and by a contract from the U.S. National Cancer Institute.

## References


Amir, E., 1999. Object-oriented first-order logic. Electronic Transactions on Artificial Intelligence 3, 63–84.

Bechhofer, S., Goble, C., Horrocks, I., 2001. DAML+OIL is not enough. In: The First Semantic Web Working Symposium. Stanford, CA.

Bernstein, P. A., Halevy, A. Y., Pottinger, R. A., 2000. Model management: Managing complex information structures. SIGMOD Record 29 (4), 55–63.

Calvanese, D., Giacomo, G., Lenzerini, M., 2001. Ontology of integration and integration of ontologies. In: Description Logic Workshop (DL 2001). pp. 10–19.

Calvanese, D., Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R., 1998a. Information integration: Conceptual modeling and reasoning support. In: 6th International Conference on Cooperative Information Systems (CoopIS'98). pp. 280–291.

Calvanese, D., Lenzerini, M., Nardi, D., 1998b. Description logics for conceptual data modeling. In: Chomicki, J., Saake, G. (Eds.), Logics for Databases and Information Systems. Kluwer Academic Publisher, pp. 229–263.

Castano, S., De Antonellis, V., De Capitani di Vemercati, S., 2001. Global viewing of heterogeneous data sources. IEEE Transactions on Data and Knowledge Engineering 13 (2), 277–297.

Chalupsky, H., 2000. OntoMorph: A translation system for symbolic knowledge. In: Cohn, A. G., Giunchiglia, F., Selman, B. (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000). Morgan Kaufmann Publishers, San Francisco, CA.

Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J., 1998. OKBC: A programmatic foundation for knowledge base interoperability. In: Fifteenth National Conference on Artificial Intelligence (AAAI-98). AAAI Press/The MIT Press, Madison, Wisconsin, pp. 600–607.

DAML, ???? DAML ontology library, http://www.daml.org/library.

Dean, M., Connolly, D., Harmelen, F. v., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., 2002. Web ontology language (OWL) reference version 1.0, http://www.w3.org/tr/owl-guide/.

Doan, A., Domingos, P., Halevy, A., 2001. Reconciling schemas of disparate data sources: A machine learning approach. In: ACM SIGMOD Conf. on Management of Data (SIGMOD-2001).

Doan, A., Madhavan, J., Domingos, P., Halevy, A., 2002. Learning to map between ontologies on the semantic web. In: The Eleventh International WWW Conference. Hawaii, US.

Dou, D., McDermott, D., Qi, P., 2002. Ontology translation by ontology merging and automated reasoning. In: EKAW'02 workshop on Ontologies for Multi-Agent Systems. Sigüenza, Spain.

Farquhar, A., Fikes, R., Rice, J., 1996. The Ontolingua server: a tool for collaborative ontology construction. In: Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.

Fellbaum, C. (Ed.), 1998. WordNet: An Electronic Lexical Database. MIT Press, Cambridge.

Fensel, D., Benjamins, V. R., Motta, E., Wielinga, R., 1999. UPML: A framework for knowledge system reuse. In: International Joint Conference on Artificial Intelligence (IJCAI-99). Stockholm, Sweden.

Fensel, D., Horrocks, I., Harmelen, F. V., Decker, S., Erdmann, M., Klein, M., 2000. OIL in a nut-

shell. In: 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000). Springer, Juan-les-Pins, France.

Franconi, E., Ng, G., 2000. The i.com tool for intelligent conceptual modelling. In: 7th Intl. Workshop on Knowledge Representation meets Databases (KRDB'00). Berlin, Germany.

Ganter, B., Wille, R., 1999. Formal Concept Analysis: Mathematical foundations. Springer, Berlin-Heidelberg.

Gennari, J., Grosso, W., Musen, M., 1998. A method-description language: An initial ontology with examples. In: Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop. Banff, Canada.

Gruber, T. R., 1993. Toward principles for the design of ontologies used for knowledge sharing. Tech. Rep. KSL 93-04, Knowledge Systems Laboratory, Stanford University.

Hendler, J., McGuinness, D. L., 2000. The DARPA agent markup language. IEEE Intelligent Systems 16 (6), 67–73.

Hovy, E., 1998. Combining and standardizing largescale, practical ontologies for machine translation and other uses. In: First International Conference on Language Resources and Evaluation (LREC). Granada, Spain, pp. 535–542.

Lenat, D. B., 1995. Cyc: A large-scale investment in knowledge infrastructure. Communications of ACM 38 (11), 33–38.

Lindberg, D., Humphreys, B., McCray, A., 1993. The unified medical language system. Methods of Information in Medicine 32 (4), 281.

Madhavan, J., Bernstein, P., Rahm, E., 2001. Generic schema matching using Cupid. In: 27th International Conf. on Very Large Data Bases (VLDB '01). Rome, Italy.

Madhavan, J., Bernstein, P. A., Domingos, P., Halevy, A., 2002. Representing and reasoning about mappings between domain models. In: Eighteenth National Conference on Artificial Intelligence (AAAI'2002). Edmonton, Canada.

McGuinness, D. L., Fikes, R., Rice, J., Wilder, S., 2000. An environment for merging and testing large ontologies. In: Cohn, A. G., Giunchiglia, F., Selman, B. (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000). Morgan Kaufmann Publishers, San Francisco, CA.

Mena, E., Illarramendi, A., Kashyap, V., Sheth, A., 2000. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distributed and Parallel Databases—An International Journal 8 (2).

Miller, R., Haas, L., Hernandez, M., 2000. Schema mapping as query discovery. In: 26th International Conference On Very Large Databases (VLDB'2000).

Mitra, P., Wiederhold, G., Decker, S., 2001. A scalable framework for interoperation of information sources. In: The 1st International Semantic Web Working Symposium (SWWS'01). Stanford University, Stanford, CA.

Modgil, S., Hammond, P., Wyatt, J., Potts, H., 2000. The design-a-trial project: Developing a knowledge-based tool for authoring clinical trial protocols. In: First European Workshop on Computer-based Support for Clinical Guidelines and Protocols (EWGLP 2000). IOS Press, Amsterdam, Leipzig, Germany.

Musen, M. A., 1992. Dimensions of knowledge sharing and reuse. Computers and Biomedical Research 25, 435–467.

Noy, N., Musen, M., 2000. PROMPT: Algorithm and tool for automated ontology merging and

alignment. In: Seventeenth National Conference on Artificial Intelligence (AAAI-2000). Austin, TX.

Noy, N. F., Musen, M. A., 2001. Anchor-PROMPT: Using non-local context for semantic matching. In: Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001). Seattle, WA.

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

Noy, N. F., Musen, M. A., J. L.V. Mejino, J., Rosse., C., 2002. Pushing the envelope: Challenges in a frame-based representation of human anatomy. Tech. Rep. SMI-2002-0925, Stanford Medical Informatics.

Palopoli, L., Sacca, D., Terracina, G., Ursino, D., 1999. A unified graph-based framework for deriving nominal interscheme properties, type conflicts and object cluster similarities. In: 4th IFCIS International Conference On Cooperative Information Systems (CoopIS). IEEE Computer, pp. 34–45.

Rahm, E., Bernstein, P. A., 2001. A survey of approaches to automatic schema matching. VLDB Journal 10 (4).

Rosse, C., Shapiro, L., Brinkley, J., 1998. The digital anatomist foundational model: Principles for defining and structuring its concept domain. Journal of American Medical Informatics Association, AMIA'98 Symposium Supplement , 820–824.

Sim, I., 1997. Trial banks: An informatics foundation for evidence-based medicine. Phd dissertation, Stanford University.

Spackman, K. (Ed.), 2000. SMOMED RT: Systematized Nomenclature of Medicine, Reference Terminology. College of American Pathologists, Northfield, IL.

Stumme, G., Mädche, A., 2001. FCA-Merge: Bottom-up merging of ontologies. In: 7th Intl. Conf. on Artificial Intelligence (IJCAI '01). Seattle, WA, pp. 225–230.

W3C, 2000. Resource description framework (RDF).