Ontology Translation by Ontology Merging and Automated Reasoning *

Dejing Dou, Drew McDermott, and Peishen Qi

Yale University, Computer Science Department
New Haven, CT 06520
{dejing.dou, drew.mcdermott, peishen.qi}@yale.edu

Abstract. Ontology translation is one of the most difficult problems that web-based agents must cope with. An *ontology* is a formal specification of a vocabulary, including axioms relating the terms. Ontology translation is best thought of in terms of ontology merging. The merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them. We add *bridging axioms* not only as "bridges" between terms in two related ontologies but also to make this merge into a complete new ontology for further merging with other ontologies. Translation is implemented using an inference engine (*OntoEngine*), running in either a demand-driven (backward-chaining) or data-driven (forward chaining) way. We illustrate our method by describing its application in an online ontology translation system, *OntoMerge*, which translates a *dataset* in the DAML notation to a new DAML dataset that captures the same information, but in a different ontology. A uniform internal representation, *Web-PDDL* is used for representing merged ontologies and datasets for automated reasoning.

1 Introduction

One of the most difficult problems that web-based agents must cope with is ontology translation, because web-based agents often use different ontologies to represent the information they want to share with each other, even when some of them use the same web-agent language such as WSDL [1] or DAML [2] to represent that information. We define an *ontology* as a formal specification of a vocabulary, including axioms relating the terms. A *dataset* is defined as a set of facts expressed using a particular ontology [24]. The ontology translation problem is to translate a dataset represented in one (source) ontology to a dataset represented in another (target) ontology. The differences between two ontologies can include syntactic and semantic differences, both of which we will have to deal with.

Previous work on ontology translation has made use of two strategies. One is to translate a dataset in any source ontology to the dataset in one big, centralized ontology that serves as an interlingua which can be translated into a

* This research was supported by DARPA as DAML program.

dataset in any target ontology. Ontolingua [20] is a typical example for this strategy, but this strategy can't really work well unless a global ontology can cover all existing ontologies, and we can get agreement by all ontology experts to write translators between their own ontologies and this global ontology. Even if in principle such harmony can be attained, in practice keeping all ontologies — including the new ones that come along every day — consistent with the One True Theory is very difficult. If someone creates a simple, lightweight ontology for a particular domain, he may be interested in translating it to neighboring domains, but can't be bothered to think about how it fits into a grand unified theory of knowledge representation. The other strategy is to do ontology translation directly from a dataset in a (source) ontology to a dataset in another (target) ontology, on a dataset-by-dataset basis, without the use of any kind of interlingua. OntoMorph [18] is a typical example of this strategy. For practical purposes this sort of program can be very useful, but it tends to rely on special properties of the datasets to be translated, and doesn't address the question of producing a general-purpose translator that handles any source dataset.

Past work [20, 18] in this area has addressed both syntactic and semantic-issues, but tends to focus more on syntactic translation [18] because it is easier to automate. The "semantic" problem is to replace the vocabulary of the source ontology with the vocabulary of the target, which is much more difficult. The reason it is so difficult is that it often requires subtle judgment about the relationships between meanings of terms in one ontology and the meanings of terms in another ontology. Semantic translation requires ontology experts' intervention and maintenance [24] and can not be fully automated because only humans can understand the meanings of terms and the relationships between these terms.

In this paper we present the theory and algorithms underlying our online ontology translation system, *OntoMerge*, which is based on a new approach to the translation problem: ontology translation by ontology merging and automated reasoning. The *merge* of two related ontologies is obtained by taking the union of the terms and the axioms defining them, using XML namespaces to avoid name clashes. We then add *bridging axioms* that relate the terms in one ontology to the terms in the other through the terms in the merge. We develop one merged ontology not only as a "bridge" between two related ontologies but also as a new ontology for further merging with other ontologies in the ontology community.

Although ontology merging requires ontology experts' intervention and maintenance, automated reasoning by an inference engine (*OntoEngine*) can be conducted in the merged ontology in either a demand-driven (backward-chaining) or data-driven (forward chaining) way to implement ontology translation automatically.

2 Our Approach

2.1 Uniform Internal Representation

As we have pointed out, past work in the area of ontology translation has usually mixed up syntactic translation and semantic translation. We think it is more enlightening to separate the two. If all ontologies and datasets can be expressed in terms of some uniform internal representation, we can focus on semantic operations involving this representation, and handle syntax by translating into and out of it. Although the users don't need to know the details of this internal representation, getting them right is important to make our program work. For web-based agents, the representation should contain the following elements:

1. A set of XML namespaces
2. A set of types related to namespaces.
3. A set of symbols, each with a namespace and a type.
4. A set of axioms involving the symbols.

Our language, called "*Web-PDDL*", is the AI plan-domain definition language PDDL augmented with XML namespaces and more flexible notations for axioms [25]. Like the original PDDL [23], Web-PDDL uses Lisp-like syntax and is a strongly typed first order logic language. Here is an example, part of a bibliography ontology [6] written in Web-PDDL. The DAML version of this ontology is at http://www.daml.org/ontologies/81.

```
(define (domain yale_bib-ont)
        (:extends (uri "http://www.w3.org/2000/01/rdf-schema#"
                   :prefix rdfs))
        (:requirement :existential-preconditions
                      :conditional-effects)
        (:types Publication - Obj
              Article Book Techreport Incollection
              Inproceedings - Publication
              Literal - @rdfs:Literal)
        (:predicates (author p - Publication a - Literal)
                .....
```

Assertions using this ontology are written in the usual Lisp style: `(author pub20 "Tom Jefferson")`, for instance. Quantifiers and other reserved words use a similar parenthesized syntax.

We write types starting with capital letters. A constant or variable is declared to be of a type T by writing "`x -` T". To make the namespace extensions work, we have broadened the syntax for expressing how one ontology ("domain") extends another. In the original PDDL, there was no specification of how domain names were associated with domain definitions. On the web, the natural way to make the association is to associate domains with URIs, so we replace simple domain names with `uri` expressions, which include a `:prefix` specification similar to XML namespace prefixes. With this new feature, we can add `@prefix:` to each term in a Web-PDDL file to declare its namespace (ontology); for instance, `@rdfs:Literal` means a type from the `rdfs` namespace. Symbols without a prefix come from the local namespace.

Types can be thought of as sets, but they are not first-class terms in the language. The sets denoted by two types must be disjoint unless one is a subtype of the other. It is sometimes useful to state that an object `x` has type T, where T is a subtype of the type it was declared with. For this purpose you use the pseudo-predicate `is`, writing `(is` T `x)`.

If someone wants to use our system with an external representation other than Web-PDDL, there must exist a translator between the two. We have provided such a translator, which we call *PDDAML* [11], for translating between Web-PDDL and DAML. Writing translators for other XML-based web languages would not be difficult. In the following sections, we will use Web-PDDL to describe our work on ontology merging and automated reasoning, and ignore the external representation.

2.2 Ontology Merging and Bridging Axioms

Once we have cleared away the syntactic differences, the ontology translation problem is just semantic translation from the internal representations of a dataset in the source ontology to the internal representations of a dataset in the target ontology. Semantic translation requires ontology experts' intervention and maintenance because only humans can understand the relationships between the meanings of terms in one ontology and terms in another ontology. For the rest of this paper, we will use two alternative bibliography ontologies as a running example. These were developed as contributions to the DAML Ontology Library, and while they are both obviously derived from Bibtex terminology, different design decisions were made by the people who derived them. One was developed at Yale, and we have given it the prefix `yale_bib` [4]; the other was developed at CMU, and it gets the prefix `cmu_bib` [5]. Although neither of them is a realistically complex ontology, the semantic differences that arise among corresponding types, predicates and axioms in these two ontologies serve as good illustrations of what happens with larger examples.

For example, both ontologies have a type called `Article`, but `@cmu_bib:Article` and `@yale_bib:Article` mean two different things. In the `yale_bib` ontology, `Article` is a type which is disjoint with other types such as `Inproceedings` and `Incollection`. Therefore, `@yale_bib:Article` only means those articles which were published in a journal, but `@cmu_bib:Article` includes all articles which were published in a journal, proceedings or collection.

Another example: both of these ontologies have a predicate called `booktitle`. In the `cmu_bib` ontology, `(booktitle ?b - Book ?bs - String)` means `?b` is a `Book` and it has title `?bs` as String. While in the `yale_bib` ontology, `(booktitle ?p - Publication ?pl - Literal)` means `?p` is an `Inproceedings` or an `Incollection`, and `?pl` is the title of the proceedings or collection which contains `?p`.

Here is how these distinctions would come up in the context of an ontology-translation problem. Suppose the source dataset uses the `yale_bib` ontology, and includes this fragment:

```
(:objects ... Jefferson76 - Inproceedings)
...
(:facts ... (booktitle Jefferson76 "Proc. Cont. Cong. 1") ...)
```

The translated dataset in the cmu_bib ontology would then have to include this:

```
(:objects ... Jefferson76 - Article
              proc301 - Proceedings)
...
(facts ... (inProceedings Jefferson76 proc301)
           (booktitle proc301 "Proc. Cont. Cong. 1") ...)
```

Note that we have had to introduce a new constant, `proc301`, to designate the proceedings that `Jefferson76` appears in. Such *skolem terms* [28] are necessary whenever the translation requires talking about an object that can't be identified with any existing object.

It is tempting to think of the translation problem as a problem of rewriting formulas from one vocabulary to another. The simplest version of this approach is to use forward-chaining rewrite rules. A more sophisticated version is to use lifting axioms [17], axioms of the form `(if p q)`, where `p` is expressed entirely in the source ontology and `q` entirely in the target ontology. Such a rule could be used in either a forward or backward (demand-driven) way, but the key improvement is that the axiom is a true statement, not an ad-hoc rule. This is essentially the idea we will use, except that we do away with any restrictions on the form or the vocabulary of the rules, and we adopt the term "bridging axiom" for them. A *bridging axiom* is then any axiom at all that relates the terms of two or more ontologies. By looking for such axioms, we separate the problem of relating two vocabularies from the problem of translating a particular dataset. This makes dataset translation a slightly more difficult problem, but the flexibility and robustness we gain are well worth it.

Another antecedent of our research is the work on data integration by the database community. For a useful survey see [21]. The work closest to ours is that of [22], who use the term "helper model" to mean approximately what we mean by "merged ontology." One difference is that in their framework the bridging rules are thought of as metarules that link pairs of data sources, whereas we embed the rules in the merged ontology. However, the greatest difference is that for databases the key concerns are that inter-schema translations preserve soundness and completeness. In the context of the Semantic Web, while soundness is important, it is not clear even what completeness would mean. So we have adopted a more empirical approach, without trying to assure that all possible useful inferences are drawn. Because it is entirely possible that an ontology could introduce undecidable inference problems, it's not clear how we could do better than that.

Bridging axioms use vocabulary items from both the source and target ontologies, and in fact may make use of new symbols. We have to put these symbols somewhere, so we introduce the concept of the *merged ontology* for a set of *component ontologies*, defined as a new ontology that contains all the symbols of the components, plus whatever new ones are needed. (Namespaces ensure that the symbols don't get mixed up.)

We can now think of dataset translation this way: Take the dataset and treat it as being in the merged ontology covering the source and target. Draw conclusions from it. The bridging axioms make it possible to draw conclusions from premises some of which come from the source and some from the target, or to draw target-vocabulary conclusions from source-language premises, or vice versa. When we get a pure conclusion in either vocabulary from merged-ontology premises we call it a *projection* into that vocabulary. The approach to translation that we focus on in this paper is to do forward inference from a source dataset in the merged vocabulary, retaining the projections as the result. In some cases, backward chaining would make more sense, as we discuss in the "Future Work" section below. In either case, the idea is to push inferences through the pattern

$$\text{source} \Leftrightarrow \text{merge} \Leftrightarrow \text{target}.$$

Getting back to our example of merging the `yale_bib` and `cmu_bib` ontologies, we suppose we are using a merged ontology with prefix `cyb_merging`. When one term (type or predicate) in the `yale_bib` ontology has no semantic difference with another term in the `cmu_bib` ontology, we just use a new term with the same meaning as the old ones, then add bridging axioms to express that these three terms are the same. It happens that (as far as we can tell), `@cmu_bib:Book` is the same type as `@yale_bib:Book`, so we can introduce a type Book in the `cyb_merging` ontology, and define it to mean the same as the other two. Because types are not objects, we cannot write an axiom such as `(= Book @cmu_bib:Book)`. (A term with no namespace prefix should be assumed to live in the merged ontology.) So we have to use a pseudo-predicate (or, perhaps, "meta-predicate") `T->` and write rules of the sort shown below. We call these *type-translation rules*:

```
(axioms:
        (T-> @cmu_bib:Book Book)
        (T-> @yale_bib:Book Book)
        ...
```

The general form of a type-translation rule is

```
(T-> type1 type2 [P])
```

which means "`type1` is equivalent to `type2`, except that objects of `type1` satisfy property `P`." We call `P` the distinguisher of `type1` and `type2`. If `P` is omitted, it defaults to true. For instance, suppose ontologies `ont-A` and `ont-B` both use the type `Animal`, but `ont-A` uses it to mean "terrestrial animal". Then the type-translation rule would be

```
(T-> @ont-A:Animal @ont-B:Animal (terrestrial x))
```

All other axioms can be stated in ordinary first-order logic. Consider the predicate title, declared as `(title ?p - Publication ?t - String)` in both the `yale_bib` and `cmu_bib` ontologies. We just reuse the same symbol in our `cyb_merging` ontology. The corresponding bridging axioms become:

```
(forall (?p - Publication ?ts - String)
        (iff (@cmu_bib:title ?p ?ts) (title ?p ?ts)))
```

```
(forall (?p - Publication ?ts - String)
        (iff (@yale_bib:title ?p ?ts) (title ?p ?ts)))
```

When one term (type or predicate) in the `yale_bib` ontology has a semantic difference from the related term in the `cmu_bib` ontology, the bridging axioms become more substantial. Sometimes the axioms reflect our decision that one of the component ontologies gets a concept closer to "right" than the other. In our example, we might decide that `cmu_bib` has a better notion of `Article` (a type). Then we add bridge axioms for `Article`, `@cmu_bib:Article`, and `@yale_bib:Article`:

```
(T-> @cmu_bib:Article Article)

(forall (?a - Article)
        (iff (exists (?j - Journal ?s - Issue)
                     (and (issue ?s ?j) (contains ?s ?a)))
             (is @yale_bib:Article ?a)))
```

The above bridging axioms mean: in `cyb_merging` ontology, `Article` is the same type as `@cmu_bib:Article`, which we prefer. But if and only if an *Article* is contained in some `Journal`, it is a `@yale_bib:Article`.

We also add bridge axioms for `booktitle`, for which we decide that `yale_bib`'s predicate makes more sense. The axioms relate `@cmu_bib:booktitle`, `@yale_bib:booktitle`, and our new predicate thus:

```
(forall (?a - Article ?tl - String)
        (iff (@yale_bib:booktitle ?a ?tl)
             (booktitle ?a ?tl)))
```

```
(forall (?a - Article ?tl - String)
        (iff (booktitle ?a ?tl)
             (exists (?b - Book)
                     (and (contains ?b ?a)
                          (@cmu_bib:booktitle ?b ?tl)))))
```

These bridging axioms mean: we make `booktitle` in the merged ontology be the same predicate as `@yale_bib:booktitle`, and then declare that, if and only if `?a` is an `Article` with `booktitle ?tl`, there exists some `Book ?b` such that `?b` contains `?a` and `?b` has `@cmu:booktitle ?tl`.

The full version of the merge of the `cmu_bib` and `yale_bib` ontologies can be found at [8].

Given the merged ontology, any term in either component ontology can be mapped into a term or terms in the `cyb_merging ontology`, some terms in `cyb_merging` can be projected back into term(s) in `cmu_bib` and some into term(s) in `yale_bib`. When some datasets represented in the `yale_bib` ontology need to be translated into the datasets represented in the `cmu_bib` ontology, an automated reasoning system, such as a theorem prover, can do inference by using those bridging axioms to implement translation, as we explained above. We will explore the forward-chaining option in depth in the next section.

We should mention one additional advantage of merged ontologies compared to direct translation rules [18, 17] between two related ontologies. The merged ontology serves not only as a "bridge" between the two given related ontologies, but also as a new ontology for further merging. If `foo1_bib`, `foo2_bib`, `foo3_bib` ... come out, we can use `cyb_merging` as a starting point for building a merged ontology that covers them all, or we may prefer a more incremental strategy where we merge `foo1_bib` and `foo2_bib`, creating, say "`foo_1_2_merging`", which would be used for translating between these two, then merge it with `cyb_merging` if and when a need arises for translations involving components of both merged ontologies. Exactly how many merged ontologies one needs, and how to select the right one given a new translation problem, are open research questions that we discuss in "Future Work" below. But the point to make here is that the number of merged ontologies one needs is unlikely to be as large as the number of rewrite-rule sets one would need under the direct-translation approach. If there are N ontologies that need to be translated into each other, the direct-translation approach requires $N(N-1)$ rule sets (assuming for simplicity that all pairwise combinations occur), which could be cut to $N(N-1)/2$ if the rules are bidirectional. Our approach requires on the order of N merged ontologies. Of course, the exact amount of work saved depends on the sizes of the overlaps among the component ontologies, and how these sizes change as they are merged, both of which are hard to predict.

2.3 Automated Reasoning

Theorem proving scares people. The desired to avoid it has led researchers to develop web languages, such as RDF and DAML, that look more like description logics [16] and not so much like predicate calculus, putting up with the awkwardness of description logics in hopes of reaping some benefit from their tractable computational properties. We reverse that decision by translating DAML back into predicate calculus and doing old-fashioned theorem proving on the resulting internal representation. Our rationale is that, based on the examples we have looked at, we are inclined to think most of the theorem-proving problems that arise during ontology translation, while not quite within the range of Datalog, are not that difficult [24]. Our inference engine, called *OntoEngine*, is a special-purpose theorem prover optimized for these sorts of problems. When some dataset in one or several source ontologies are input, OntoEngine can do inference in the merged ontology, collecting the resulting projections into one or several target ontologies automatically.

A key component of OntoEngine is the indexing structure that allows it to find formulas to be used in inference. Figure 1 shows what this structure looks like. As ontologies and datasets are loaded into OntoEngine, their contents are stored into this structure, as are the results of inferences.

We use a combination of tree-based and table-based indexing. At the top level we discriminate on the basis of namespace prefixes. For each namespace, there is an *ontology node.* The facts stored in an ontology node are next discriminated by predicate. The resulting *Predicate node*s are then discriminated into five categories:

1. Predicate declarations
2. Positive literals (atomic formulas)
3. Negative literals (negated atomic formulas)
4. Implications with the predicate in the premise
5. Implications with the predicate in the conclusion

Predicate declarations are expressions such as `(title ?p - Publication ?tl - String)`. Positive literals and negative literals are facts such as `(title b1 "Robo Sapiens")` and `(not (title b1 "John Nash"))`.

All other formulas from bridging axioms except type translation rules can be expressed as *implications* in INF (Implicative Normal Form):

$$P_1 \wedge \cdots \wedge P_j \cdots \wedge P_n \Rightarrow Q_1 \wedge \cdots \wedge Q_k \wedge \cdots \wedge Q_m$$

where each P_i and Q_j is an atomic formula. We use the word *clause* as a synonym for *implication.* The conjunction of P's we call the *premise* of the clause; the conjunction of Qs we call the *conclusion.*

In the indexing structure, the fourth and fifth categories are implications in which the predicate in question occurs in the premise or conclusion, respectively. (Of course, it could occur in both, in which case the formula would be indexed into both categories.)

So far, we have implemented a forward-chaining inference algorithm using the indexing structure of figure 1. The algorithm is shown in figure 2. In the procedure `Forwardchaining`, the phrase "best clause" needs some explanation. Theoretically, since we are drawing all possible inferences, it doesn't matter in

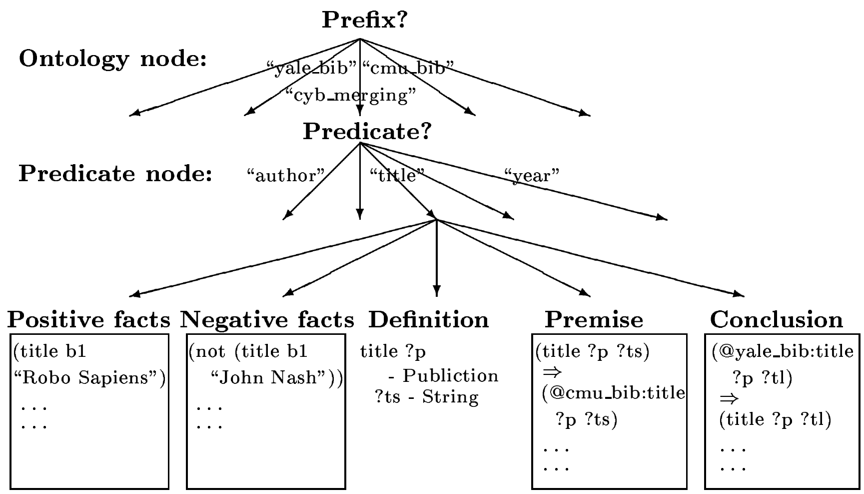

Fig. 1. Indexing Structure for OntoEngine

what order we draw them. However, in our current version of the algorithm there are cases where doing the inferences in the wrong order would result in incompleteness. If clause 1 is

$$P_1 \Rightarrow Q_1 \wedge Q_2$$

and clause 2 is

$$P_2 \wedge Q_2 \Rightarrow Q_3$$

then our algorithm will fail to conclude Q_3 from P_1 and P_2 unless clause 1 runs first, because Q_2 has to be present for clause 2 to conclude Q_3. To compensate, we use a heuristic to try to ensure that we get as many conclusions as possible as early as possible. The heuristic is to choose the "best clause," defined as a weighted average:

$$W_1 \times \text{size of conclusion} - W_2 \times \text{size of premise}$$

The design of OntoEngine profited from our study of KSL's JTP (Java Theorem Prover [9]). In the end we decided not to use JTP, but to develop our own prover, because JTP didn't contain some of the mechanisms we needed, especially type-constrained unification, while at the same time being oriented too strongly toward the traditional theorem-proving task.

3 Application: OntoMerge

We have embedded our deductive engine in an online ontology-merging service called *OntoMerge*[10]. In addition to OntoEngine, OntoMerge uses PDDAML[11] to translate into and out of the knowledge-representation language DAML+OIL[3].

```
Procedure Process(facts)
    Repeat
        oneFact = next fact in facts
        Forwardchaining(oneFact)
    Until last fact in facts
```

```
Procedure Forwardchaining(fact)
    Get best INF clause from corresponding Premises
    newFacts = Modus_Ponens(fact, clause)
    Repeat
        newFact = next fact in newFacts
        Forwardchaining(newFact)
    Until last fact in newFacts
```

```
Function Modus_Ponens(fact, clause) return facts
    Repeat
        oneAtf = next AtomicFormula from leftside of clause
        substi = Unify(oneAtf, fact)
        Add substi into the whole substitutions and check its consistency
    Until get the consistent whole substitutions
    Get newFacts by substituting rightside of clause with substitutions
    Repeat
        newFact = next fact in newFacts
        If newFact belongs to target ontology Then store it
        Else add it into facts which will be returned for further inference
    Until last fact in newFacts
    Return facts
```

```
Function Unify(oneAtf, fact) return substitutions
    Repeat
        Get one variable from oneAtf and corresponding constant from fact
        Typecheck(variable, constant)
        If succeed Then make single substitution and add it to substitutions
    Until there is no more variable
    Return substitutions
```

```
Function Typecheck(variable, constant) return boolean
    match = false
    If variable's type is same as or the super type of constant's type Then match = true
    Else
        typeCons = type of constant
        fact = (is typeCons constant)
        Forwardchaining(fact)
        If exist var, (is typeCons var) is the only premise of some clause and
          variable's type is same as or the super type of var's type Then match = true
    Return match
```

Fig. 2. The forward-chaining algorithm

OntoMerge serves as a semi-automated nexus for agents and humans to find ways of coping with notational differences, both syntactic and semantic, between ontologies. It accepts a dataset as a DAML file in the source ontology, and will respond with the dataset represented in the target ontology, also as a DAML file.

When receiving a DAML file, OntoMerge calls PDDAML to translate it into a translation problem expressed as a Web-PDDL file which is input to online version of OntoEngine. To do anything useful, OntoEngine needs to retrieve a merged ontology from its library that covers the source and target ontologies. Such an ontology must be generated by human experts, and if no one has thought about this particular source/target pair, before, all OntoEngine can do is record the need for a new merger. (If enough such requests come in, the ontology experts may wake up and get to work.) Assuming a merged ontology exists, located typically at some URL, OntoEngine tries to load it in. If it is written in DAML or other Web language, OntoEngine first calls PDDAML to translate it to Web-PDDL file, then loads it in. Finally, OntoEngine loads the dataset(facts) which need to be translated in and processes those facts one by one until no new fact is left nor generated. To get a final output file in DAML, OntoMerge calls PDDAML again to translate the Web-PDDL dataset to the corresponding dataset in DAML.

OntoMerge has worked well so far, although our experience is inevitably limited by the demand for our services. In addition to the toy example from the dataset in the `yale_bib` ontology to the dataset in the `cmu_bib` ontology, we have also run it to translate a dataset with more than 2300 facts about military information of Afghanistan using more than 10 ontologies into a dataset in the `map` ontology [14]. About 900 facts are related to the geographic features of Afghanistan in the `geonames` ontology [12] and its airports in the `airport` ontology [13]. We have merged the `geonames` ontology and the `airport` ontology with the `map` ontology. After OntoEngine loads the two merged ontologies in, it can accept all 2300 facts and translate those 900 facts in the `geonames` and `airport` ontologies into about 450 facts in the `map` ontology in 7 minitues. For each `@geonames:Feature` or each `@airport:Airport`, the bridging axioms in the merged ontologies will be used for inference to create a pair of skolem terms with types `@map:Point` and `@map:Location` in the fact like (`@map:location` *Location01 Point02*). The values of the `@geonames:longitude` (`@airport:longitude`) property and the `@geonames:latitude` (`@airport:latitude`) property for each `@geonames:Feature` (`@airport:Airport`) can be translated into the values of the `@map:longitude` property and the `@map:latitude` property for the corresponding `@map:Location`. The value of the `@airport:icaoCode` property for each `@airport:Airport` and the value of `@geonames:uniqueIdentifier` property for each `@geonames:Feature` can be translated into the values of `@map:label` property for the corresponding `@map:Point`. The reason that the translated dataset only has 450 facts is some facts in the `geonames` and `airport` ontologies can't be translated to any term in the the `map` ontology.

Prospective users should check out the OntoMerge website[1]. The website is designed to solicit descriptions of ontology-translation problems, even when OntoMerge can't solve them. However, we believe that in most cases we can develop a merged ontology within days that will translate any dataset from one of the ontologies in the merged set to another.

4 Future Work

4.1 Backward Chaining

Although so far forward-chaining deduction has been sufficient for our needs, we recognize that backward chaining is also necessary. For example, suppose one agent has a query:

```
(and (father Fred ?x) (travel ?x ?y) (desti ?y "SF"))
```

which means "Did Fred's father travel to South Florida?", and this query could be answered by another agent with its datasets in another ontology, which may have different meanings for `travel` or `desti`. In this case, the ontology-translation problem becomes the problem of answering the query in the target ontology with the datasets in the source ontology.

In addition, backward chaining may be necessary in the middle of foward chaining. For example, when OntoEngine is unifying the fact `(P c1)` with `(P ?x)` in the axiom:

$$(\texttt{P ?x}) \wedge (\texttt{member ?x } [\texttt{c1}, \texttt{c2}, \texttt{c3}]) \Rightarrow (\texttt{Q ?x})$$

it can't conclude `(Q c1)` unless it can verify that `c1` is a member of the list `[c1,c2,c3]`, and the only way to implement this deduction is by doing backward chaining.

A full treatment of backward chaining across ontologies would raise the issue of *query optimization*, which we have not focused on yet. There is a lot of work in this area, and we will cite just two references: [19, 15]. We don't yet know how important this issue will be in the context of datasets found on the Web.

4.2 Methodology and Semi-automatic Tools for Ontology Merging

Although really knotty ontology-merging issues will have to be solved by applying human insight to the production of bridging axioms, we believe it may be possible to develop automated tools to help in the merging process.

We said that the merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them. We then add bridging axioms that relate the terms in one ontology to the terms in the other through the terms

[1] http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html

in the merge. This is rather different from what some other people have emphasizedin talking ontology combination. For example, in the PROMPT [27] and Chimaera [26] systems, the focus is on ontology editing for two similar ontologies. They try to do ontology matching semi-automatically but they basically provide user with some suggestions for merging similar terms, almost always because of name similarity. The reason why it is hard to do anything else in most ontology-matching systems is that most current ontologies contain more types(i.e., classes) and predicates than axioms which relate these types and predicates. Many ontologies contain no axioms at all, even when it is obvious that their designers know more about the domain than subclass relationships.

We expect that as ontology designers become more sophisticated, they will want, nay, will demand, the ability to include more axioms in what they design. This trend may support the development of real semi-automatic tools to do ontology merging. These tools will go beyond name similarity, and try to find matches between ontologies that preserve the truth of the axioms on either side. For example, "cop" is superficially similar to "cap," but any attempt to view them as similar will cause the axioms involving them to become untranslatable.

4.3 The Ontology Covering Problem

Another interesting issue in ontology merging is the "ontology covering problem". Given a set of source and target ontologies, OntoMerge looks for a merged ontology that was formed by merging exactly the set of ontologies it's now looking at. But in general it might not find such an ontology in its database. Instead it may find one that covers a subset of them, another that covers a superset, another that covers a set that intersects the set we're looking for. Here we introduce the term *covers*, defined thus: Ontology "covers" other ontologies $O_1, \ldots, O_k$ if it is the merger of a superset of $O_1, \ldots, O_k$. It covers $O_1, \ldots, O_k$ exactly if it is the merger of $O_1, \ldots, O_k$.

Here is an example. Suppose OntoMerge is given two source ontologies Onto_1 and Onto_2, and a target ontology Onto_3. It would prefer to do all reasoning in a merged ontology that covers these three exactly. If it has never seen any of these ontologies before, it must report failure, and ask the knowledge engineers to merge them. But as more and more ontologies are developed by the experts, it's possible that OntoMerge will have in its database *Onto_1_3_5*, the merger of Onto_1, Onto_3, and Onto_5, and *Onto_2_4_5*, the merger of Onto_2, Onto_4, and Onto_5. Now it has a choice of approaches:

- Run inferences in the combination of Onto_1_3_5 and Onto_2.
- Run inferences in the combination of Onto_1_3_5 and Onto_2_4_5.
- Stop and ask the experts to merge Onto_1_3_5 and Onto_2_4_5.
- Stop and ask the experts to merge Onto_1, Onto_2, and Onto_3.

(Here, "combination" is just the union of two ontologies with no further bridging axioms involved.)

It is not obvious which is the correct choice. We would at least like it to be the case that the worst consequence of an incorrect choice is that some inferences may be missed, or that the deduction process runs with less than optimal efficiency. But we don't know how to guarantee these properties.

Or suppose the ontology DB contains one ontology covering ontologies 1, 2, 3, and 4, and another covering 1, 2, 3, and 5. On paper, either will work. But will we get the same inferences (projecting the vocabulary of Onto_4 and Onto_5 away)? Is one more efficient?

5 Conclusions

We have described a new approach to implement ontology translation by ontology merging and automated reasoning. Here are the main points we tried to make:

1. Ontology translation is best thought of in terms of *ontology merging.* The merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them, then adding bridging axioms that relate the terms in one ontology to the terms in the other through the terms in the merge.
2. It is important to separate syntactic translation and semantic translation. If all ontologies and datasets can be expressed in terms of some uniform internal representation, semantic translation can be implemented by automatic reasoning. Web-PDDL is an ideal uniform internal representation; the syntactic translation can be done by dialect-dependent modules (such as PDDAML) between Web-PDDL and other Web agent language (such as DAML).
3. We have developed a general-purpose inference system, OntoEngine, for performing automated reasoning in merged ontologies for the purpose of ontology translation. The key features of OntoEngine are its indexing structures for managing multiple ontologies, control rules for ordering forward-chaining operations, and the use of type-constrained unification.
4. We set up an ontology translation server, OntoMerge, to apply and validate our method. We hope OntoMerge can attract more ontology translation problem from other people and get their feedback, which will help our future work.

References

1. http://www.w3c.org/TR/wsdl
2. http://www.daml.org/
3. http://www.daml.org/2001/03/daml+oil-index.html
4. http://www.cs.yale.edu/homes/dvm/daml/ontologies/daml/yale_bib.daml
5. http://www.daml.ri.cmu.edu/ont/homework/atlas-publications.daml
6. http://cs-www.cs.yale.edu/homes/ddj/ontologies/yale_bib_ont.pddl
7. http://cs-www.cs.yale.edu/homes/ddj/ontologies/cmu_atlas_publications.pddl
8. http://cs-www.cs.yale.edu/homes/dvm/daml/ontologies/pddl/cmu_yale_bib_merging.pddl
9. http://www.ksl.stanford.edu/software/JTP/
10. http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html

11. http://cs-www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator.html
12. http://www.daml.org/2002/04/geonames/geonames-ont.daml
13. http://www.daml.org/2001/10/html/airport-ont.daml
14. http://www.daml.org/2001/06/map/map-ont.daml
15. S. Adali, K. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 137–148, 1996.
16. F. Baader, D. McGuinness, D. Nardi, and P. P. Schneider 2002 *The Description Logic Handbook.* Cambridge University Press
17. Sasa Buvac and Richard Fikes. A declarative formalization of knowledge translation. In *Proceedings of the ACM CIKM: The 4th International Conference on Information and Knowledge Management*, 1995.
18. Hans Chalupsky. Ontomorph: A translation system for symbolic logic. In *Proc. Int'l. Con. on Principles of Knowledge Representation and Reasoning*, pages 471–482, 2000. San Francisco: Morgan Kaufmann.
19. Michael R. Genesereth, A. Keller, and O.M. Duschka. Infomaster: An information integration system. In *Proc 97 ACM SIGMOD International Conference on Management of Data*, pages 539–542, 1997.
20. Thomas Gruber. Ontolingua: A Translation Approach to Providing Portable Ontology Specifications. *Knowledge Acquisition* , 5(2):199–200, 1993.
21. Alon Y. Halevy. Answering queries using views: A survey. *VLDB J* , 10(4):270–294, 2001.
22. Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Halevy. Representing and Reasoning about Mappings between Domain Models. In *Proc. AAAI 2002*, 2002.
23. Drew McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003).
24. D. McDermott, M. Burstein, and D. Smith. Overcoming Ontology Mismatches in Transactions with Self-Describing Agents. In *Proceedings of Semantic Web Working Symposium*, pages 285–302, 2001.
25. Drew McDermott and Dejing Dou. Representing Disjunction and Quantifiers in RDF. In *Proceedings of International Semantic Web Conference 2002*, pages 250–263, 2002.
26. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado, April, 2000.
27. N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX.
28. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc. 1995.