Ontology Translation by Ontology Merging and Automated Reasoning

Dejing Dou, Drew McDermott and Peishen Qi

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 its 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) mode. 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

"Semantic interoperability" is the problem of achieving communication between two agents that work in overlapping domains, even if they use different notations and vocabularies to describe them. We follow common practice in using the word *ontology* as a formal specification of a vocabulary, including axioms relating its terms. *Ontology translation* is a key element of the semantic-operability problem; we define it as the problem of translating datasets, queries, or theories expressed using one ontology into the vocabulary supported by another. As web-based agents rely more and more on logical notations for communication and reasoning, the problem of ontology translation will become more important. In this paper, we argue that the problem can be thought of as a deduction task, which can be solved

This research was supported by the DARPA/DAML program.

by relatively straightforward theorem-proving techniques. However, the deduction must make use of axioms supplied by human experts. We are developing tools for making it easier for experts to create these axioms, but, for the foreseeable future, there is no way to eliminate humans from the loop entirely.

Previous work on ontology translation has made use of two strategies. One is to translate a dataset (i.e., a collection of facts) to a vocabulary associated with one big, centralized ontology that serves as an interlingua. Ontolingua [29] is a typical example of this strategy. The problem with this idea is its assumption that there can be a global ontology covering all existing ontologies. Creating such a thing would require agreement among all ontology experts to write translators between their own ontologies and the global one. 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 would be 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 [23] is a typical example of this strategy. XSLT, the XML Style Language Transformations system [27], is often used to write such translators. 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 dataset in a given notation.

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 mode (backward-chaining) or a data-driven (forward chaining) mode.

This paper is organized as follows: Section 2 gives a detailed description of our approach to ontology translation, with the focus on forward chaining. Section 3 presents OntoMerge, an online ontology translation service and its application to a real-world translation problem. Section 4 describes our recent work on backward chaining and on semiautomatic merging tools for generating the bridging axioms. Section 6 concludes the paper.

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 [38]. Like the original PDDL [36], 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.

```
(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)
                .....
```

The DAML version of this ontology is at

`http://www.daml.org/ontologies/81`.

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. Types are associated with expressions as well as with objects, so that when a domain definition is loaded into memory the Web-PDDL system can check whether an expression's type is incompatible with its syntactic context. Catching such type errors early makes debugging knowledge bases much easier. Types cannot in general be identified with "classes" as that term is used in Owl and other description logics [19]. For one thing, to make type checking feasible, the sets denoted by two types must be disjoint unless one is a subtype of the other. Nonetheless, in this paper we assume that classes and types are equivalent, leaving until a later date the issue of when classes must be treated as unary predicates instead of types.

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 one can 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 *PDDOWL*[1] [12], 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 representation of a dataset in the source ontology to the internal representation of a dataset in the target 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,

[1] Formerly called PDDAML

`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. By contrast, `@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 that `?b` is a `Book` and it has title `?bs` as String. 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* [41] 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 [22], axioms of the form `(if p q)`, where `p` is expressed entirely in the source ontology and `q` entirely in the target ontology. Such an axiom could be used in either a forward or backward (demand-driven) way, but the key improvement is that the axiom is a *statement* that is true or false (hopefully true), 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 [31]. The work closest to ours is that of [33], 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. The inference process stops with conclusions whose symbols come entirely from the target vocabulary; we call these *target conclusions*. Other conclusions are used for further inference. In the end, only the target conclusions are retained; we call this *projecting* the conclusions into the target ontology. In some cases, backward chaining would be more economical than the forward-chain/project process, as we discuss in Section 4. 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_merge`. 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_merge` 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_merge` 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-> @yale_bib:Article Article
     (exists (?j - Journal ?s - Issue)
        (and (issue ?s ?j)
             (contains ?s x))))
```

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

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)))))
```

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_merge ontology`, some terms in `cyb_merge` can be projected back into terms in `cmu_bib` and some into terms 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 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_merge` 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_merge`," which would be used for translating between these two, then merge it with `cyb_merge` 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. 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

It is common assumed that theorem proving will always lead to combinatorial explosions. The desire to avoid it has led researchers to develop web languages, such as RDF and DAML, that look more like description logics [19] 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 [44, 45], are not that difficult [37]. 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, projecting the resulting conclusions 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 are expressed as *implications* in INF (Implicative Normal Form):

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

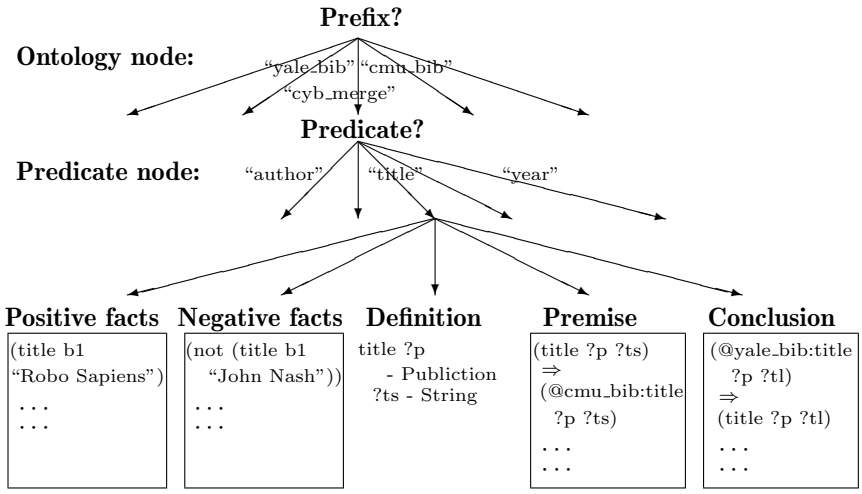

FIGURE 1. Indexing Structure for OntoEngine

where each P_i and Q_j is an atomic formula.[2] 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 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,

[2] An INF formula is equivalent to a set of one or more Horn clauses. A Horn clause is a disjunction of literals containing at most one positive literal [41]. Not all axioms can be expressed as Horn clauses, obviously, but it has been our empirical observation that bridging axioms mapping formulas to disjunctions "do not occur in nature." To some extent, this is because existing ontologies are written using relatively inexpressive languages, which leaves us in a slightly embarrassing position: If our advice that more expressive languages be used for ontologies is followed, then our current technique for doing ontology translation may turn out to be too weak.

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 profitted from our study of KSL's JTP (Java Theorem Prover [10]). 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 translation service called *OntoMerge* [11]. In addition to OntoEngine, OntoMerge uses PDDOWL [12] to translate into and out of the knowledge-representation language OWL [3]. 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 PDDOWL 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 PDDOWL to translate it to Web-PDDL file, then loads it in. Finally, OntoEngine loads the dataset which needs to be translated in and processes those facts one by one until all possible inferences have been generated. To get a final output file in DAML, OntoMerge calls PDDOWL again to translate back to DAML the projection of the Web-PDDL dataset to the target vocabulary.

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 [15]. About 900 facts are related to the geographic features of Afghanistan in the `geonames` ontology [13] and its airports in the `airport` ontology [14]. We have merged the `geonames` ontology and the `airport` ontology with the `map` ontology. After OntoEngine loads the two merged ontologies

```
Procedure Process(facts)
    For each oneFact in facts Do
        Forwardchaining(oneFact)

Procedure Forwardchaining(fact)
    For each INF clause from corresponding Premise table in best-first order
        newFacts = Modus_Ponens(fact, clause)
        For each newFact in newFacts Do Forwardchaining(newFact)

Function Modus_Ponens(fact, clause) returns facts
    facts= empty set
    oneAtf = the corresponding AtomicFormula for fact in left-side of clause
    restAtfs = the remaining AtomicFormulas in left-side of clause
    substi = Unify(oneAtf, fact)
    If substi is not empty Then substitutions = Conj_match(substi, restAtfs)
    If substitutions is not empty
    Then newFacts = substituting right-side of clause with substitutions
    For each newFact in newFacts Do
        If newFact belongs to target ontology Then store it
        Else add it into facts which will be returned for further inference
    Return facts

Function Conj_match(substi, Atfs) return substitutions
    substitutions = substi
    If Atfs is empty Then return substitutions
    Get newAtfs by substituting Atfs with substi
    facts = the corresponding facts for FIRST (newAtfs)
    For each oneFact in facts do
        oneSubsti = Unify(FIRST(newAtfs), oneFact)
        If oneSubsti is not empty
        Then oneSubstis = Conj_match(oneSubsti, REST(newAtfs))
                add oneSubstis into substis
    If substis is not empty Then add substis into substitutions
    Else substitutions = empty set
    Return substitutions
```

FIGURE 2. The forward-chaining algorithm

```
Function Unify(oneAtf, fact) return substitutions
    For each variable from oneAtf and its corresponding constant from fact Do
        If Typecheck(variable, constant)
        Then add {variable/constant} to substitutions
    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
```

FIGURE 3. The forward-chaining algorithm (cont'd)

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 5 seconds [3]. 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.

[3] After the submission of the original paper to EKAW02 OMAS workshop, we have improved the performance of OntoEngine. This performance is an order of magnitude better than that originally reported.

Prospective users should check out the OntoMerge website[4]. 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. Recent 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?" 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 forward chaining. For example, when OntoEngine is unifying the fact `(P c1)` with `(P ?x)` in the axiom:

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

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.

We have embedded a backward-chaining reasoner into OntoEngine for querying through different ontologies. We have given a detailed example in [25] of reasoning using two different genealogy ontologies, one developed by Dynamics Research Corporation (DRC) [17] and one developed by BBN technologies [16]. The dataset is a set of several hundred facts expressed in the BBN genealogy; a typical query is "Who was the queen of Henry VI and what is date of their marriage?," and is expressed in the DRC genealogy,

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: [28, 18]. We intend to focus more on overcoming complicated semantic differences when querying through different ontologies.

4.2. Semiautomatic Tools for Ontology Merging

Our merging 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 in the merge.

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

Semi-Automatic tools for generating bridging axioms. To generate these bridging axioms, we must first find out the correspondence between the concepts of two ontologies, which is the target of ontology mapping. In many cases, only humans can understand the complicated relationships that can hold between the mapped concepts. Generating these axioms must involve participation from humans, especially domain experts. You can't write bridging axioms between two medical-informatics ontologies without help from biologists.

The generation of an axiom will often be an interactive process. Domain experts keep on editing the axiom till they are satisfied with the relation expressed by it. Unfortunately, domain experts are usually not very good at the formal logic syntax that we use for the axioms. It is necessary for the axiom-generating tool to hide the logic behind the scenes whenever possible. Then domain experts can check and revise the axioms using the formalism they are familiar with, or even using natural-language expressions. So instead of seeing a logical fact representation such as

```
(booktitle Jefferson76 "Proc. Cont. Cong. 1")
```

they will see a sentence like: "The title of the book containing Jefferson76 is 'Proc. Cont. Cong. 1'," assuming that there is a natural-language template associated with the predicate `booktitle`:

```
(booktitle ?pub - Publication ?lit - String) =>
   ("the title of the book containing "
      (:np "publication" ?pub)
      "is " not?
      (:select ((is-var ?lit) => ("the string " ?lit))
               (=> "'" ?lit "'"))))
```

in which `:np` starts a noun phrase and `(is-var ?var)` checks whether `?var` is bound to a variable or a non-variable. The elements in the template are delimited by whitespace and parenthesis. The whole NL expression is the concatenation of the strings generated from each elements. It is not hard for domain experts to produce this kind of template for the predicates in their domain. The depth of the coverage is up to them. Any symbols without templates will be processed using default phrases.

We have developed a semi-automatic mapping tool and a template-based natural language generator, and embedded them into our axiom-generating system. Domain experts now can generate the INF format bridging axioms as specified in Section 2.3.

The following example shows how we generate the bridging axioms between the two `booktitle` predicates in the bibliography ontologies. The mapping tool can easily suggest a mapping between these two properties because they share the same name. Domain experts are aware that the semantic meaning of yale_bib's `booktitle` conforms to that of BibTex, and prefer it be copied into the merged ontology directly. However they also know that the semantic meaning of cmu_bib's `booktitle` is not quite the same. The suggested mapping can be corrected by

adding another predicate `inProceedings` (or `inCollection`) to the cmu_bib side, along with appropriate NL templates. Then the domain experts can fill in the underlined slots with exemplar instances in the following two groups of natural language expressions, and try to make the whole expression denote the correct meaning.

> "the title of the book containing a publication is the string ?lit" ⟵ @yale_bib:booktitle
>
> **implies**
>
> "the title of a publication (a book) is the string ?lit" ⟵ @cmu_bib:booktitle
> (and | or | implies | implied by)
> 'a publication appears in the Proceedings ?proc" ⟵ @cmu_bib:inProceedings

An acceptable fill-in might be:

> "the title of the book containing Jefferson76 is the string 'Proc. Cont. Cong. 1'"
>
> **implies**
>
> "the title of Proc11 (a book) is the string 'Proc. Cont. Cong. 1'"
> and
> "Jefferson76 appears in Proc11"

Our axiom-generation system then tries to derive the axiom by generalization. It is not always simply a matter of replacing all the constants with variables and slapping universal quantifiers over them. In the example above, Proc11 shows up only on the right-hand side of the implication. It's possible that "Proc11" is a special constant that is part of the meaning of "booktitle." (The translation of `(chinese Lao)` in one ontology might be `(citizen Lao china)`. This indicates that the translation of `(chinese ?x)` is `(citizen ?x china)`, because `china` is a special constant that is part of the translation of `chinese`.) But in fact in deriving the proper axiom for translating `booktitle`, "Proc11" should be turned into an existential variable. Such choices must be made by the human users.

Consistency checking of the generated axioms will also alleviate the burdens on domain experts. For instance, in the final correct axiom, the type of the variable which takes the place of Jefferson76 should be narrowed down to `@yale_bib:InProceedings` from the much broader concept `Publication`, otherwise inconsistency would come up because the `cmu_bib:inProceedings` predicate won't hold for a `yale_bib:InCollection`.

5. Related work

Lots of research has been done on ontology mapping, including implemented systems named CUPID [34], GLUE [24] and CTX-Match [42]. Some existing systems do not recognize an explicit boundary between ontology mapping and ontology merging; these include Chimaera [39], PROMPT [40], FCA-Merge [43], and MOMIS [21]. The matching algorithms they used can be divided into two ways: ontology-based and instance-based. Instance-based approaches mainly exploit machine learning techniques to uncover the sematic relations among the concepts. Examples include GLUE and FCA-Merge. However, the situation we met in our ontology translation problem is that usually we just have two ontologies and data available in only one of them. Ontology-based approaches should be more applicable to such situations. Three levels of knowledge are utilized in locating the mappings: syntactic, structural and semantic matching. Syntactic matching focuses on the name similarity between nodes' labels, as in CUPID. Structural matching analysis the neighborhood of ontology nodes, as in PROMPT and Chimaera. Semantic matching tries to find the mapping between meanings of the concepts. Additional knowledge information like a pre-compiled thesaurus or WordNet [26] are usually used to find semantic mappings, as in CTX-Match.

However, most of the mapping tools can only find mappings with "subclassOf", "subpropertyOf" or "equivalent" relationships. The reason why it is hard to do anything better 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. Axiom-driven matching tools will go beyond those simple relationships, and try to find matches between ontologies that preserve the truth of the axioms on either side. For example, the two `booktitle` predicates in the yale_bib and cmu_bib bibliography ontologies share the same name, but any attempt to view one of them as a subproperty of the other will cause the axioms involving them to become untranslatable.

Ontology-mapping tools typically return a list of pairs of concepts, one from each source ontology.. The merging tools do return the merged ontology, but most of them consider merging only classes. All the properties in the original ontologies are copied into the merged one, unless they mean exactly the same thing. So a dataset in the source ontology can be transferred to the merged ontology, but still not in a form understandable by an agent using the target ontology. To solve the ontology-translation problem, we need a merged ontology in which the mappings are expressed as a set of bridging axioms which explain how those concepts in the original ontologies are related.

An approach similar to ours is presented in [35]; their MAFRA framework uses a semantic bridging ontology (SBO) to encode the mappings. An instance of SBO contains axiomatic rules that transform instances of a source entity to

instances of target entity. This is not quite as general as our axioms, but in practice the difference is probably of little importance.

We should also compare our system with other programs for doing inference, rule-based or otherwise. There are by now an abundance of rule languages for manipulating RDF. Any query system can be made into a rule system by providing a notation for attaching an action to a query, such that the action is performed for each instance of the query retrieved, and by now there are dozens of query systems for RDF. (A good survey is [30].) Many of these systems could be adapted to suit our purposes. Indeed, OntoMerge incorporates some infrastructure code from the open-source Jena RDF-inference toolkit [9], and we could no doubt improve the performance of OntoMerge by borrowing further optimization techniques from other sources. However, no pre-existing inference engine did everything we needed our engine to do. In particular, most rule-based inference software does not do equality substitution, nor are existing packages sensitive to the goal of drawing inferences in a target ontology. Including these facilities means tinkering with the innards of an inference engine; it would be impossible or extremely awkward to use someone else's inference system as a black box whose output we filter.

The dominant pattern for inference on the semantic web involves description-logic notations such as OWL [19, 3]. There are several efficient inference systems available for such logics [32], many of which have been equipped with front ends for RDF. The term *description logic (DL)* covers a family of subsets of first-order logic with the following features:

- All predicates are unary or binary. Unary predicates are called *classes*, and binary predicates are called *roles*. The name "class" is used for unary predicates because it is often thought of as the set of all entities the predicate is true of.
- Class terms can be created by set-theoretic operations on existing classes, plus the use of *restrictions* such as $\exists role.class$, which is the class of objects x such that there is a y such that $class(y)$ and $role(x, y)$. E.g.,

  $$\texttt{(Person} \sqcap \exists \texttt{ has-child.Female)(Fred)}$$

  asserts that `Fred` is a person who has a female child.
- Role terms can be created (in some DLs) by taking inverses ($\cdot^{-1}$) and compositions of roles.
- Some DLs extend the basic framework by allowing axioms asserting that one class is a subset of or equal to another. Some allow classes to be recursively defined.
- The standard reasoning tasks performed by DL systems include
  - *classification*: determining whether an individual belongs to a class.
  - *subsumption*: determining whether one class is a subset of another.

One reason DLs have attracted a lot of attention is that for many of them the classification and subsumption tests are decidable. Variations in the expressiveness correspond to variations in computational complexity. However, DLs can't approach the expressiveness of first-order logic without giving up decidability and

all those complexity analyses. The question, therefore, is how much expressiveness is needed for the ontology-translation task.

Unfortunately, there are two reasons why DLs fall short. One is that they lack a robust notion of forward chaining. They can be used for answering queries in merged ontologies, but not for translating datasets.

However, the more fundamental reason for the inadequacy of DLs is that even simple bridging axioms are difficult to express. Consider this axiom from section 2.2:

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

Even though the axiom uses only types (a special case of classes) and binary predicates, it is impossible to express it as a standard DL axiom. The best we can do is something like this:

$$(\exists \texttt{booktitle}^{-1}.\texttt{Article}) = \exists \texttt{@cmu\_bib:booktitle}^{-1}.\exists \texttt{contains}.\texttt{Article}$$

In English: "The strings that are booktitles of articles (in the merged ontology) are the strings that are booktitles (in the CMU ontology) of books that contain articles." This formula falls short because it doesn't *equate* the article corresponding to an article title on the left side of the axiom with the article corresponding to a containing book on the right side. To state that, we would need to use a very expressive DL that allowed equations of role compositions:

$$\texttt{booktitle} = \texttt{contains}^{-1} \circ \texttt{@cmu\_bib:booktitle}$$

However, in conjunction with other required expressivity enhancements, adding role-composition equations makes DLs undecidable [20], so we end up with a clumsy notation that may be no more efficient than first-order logic.

6. 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. For this to be feasible, the internal representation must be as flexible as possible. We believe that the language we use, Web-PDDL, has about the right degree of flexibility. Translating from other notations to Web-PDDL is performed by dialect-dependent modules. An example is our PDDOWL system, which transforms formulas in Web-PDDL to DAML, and back.

3. We have developed a special-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, its ability to do equality substitution, its mechanism for handling existential variables and skolem terms, control rules for ordering both forward and backward 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.
5. We designed a semi-automatic tool which can help generate the bridge axioms to merge ontologies. It provides a natural-language interface for domain experts, who are usually not good at logic formalism, to construct and edit the axioms.

References

[1] http://www.w3c.org/TR/wsdl
[2] http://www.daml.org
[3] http://www.w3.org/TR/owl-guide
[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_merge.pddl
[9] http://jena.sourceforge.net/tutorial/RDF_API/index.html
[10] http://www.ksl.stanford.edu/software/JTP
[11] http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
[12] http://cs-www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator.html
[13] http://www.daml.org/2002/04/geonames/geonames-ont.daml
[14] http://www.daml.org/2001/10/html/airport-ont.daml
[15] http://www.daml.org/2001/06/map/map-ont.daml
[16] http://www.daml.org/2001/01/gedcom/gedcom.daml
[17] http://orlando.drc.com/daml/Ontology/Genealogy/3.1/Gentology-ont.daml

[18] 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.

[19] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook; Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[20] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook; Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[21] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The MOMIS approach to information integration. In *ICEIS (1)*, pages 194–198, 2001.

[22] S. Buvac and R. Fikes. A declarative formalization of knowledge translation. In *Proceedings of the ACM CIKM: The 4th International Conference on Information and Knowledge Management*, 1995.

[23] H. 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.

[24] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the World-Wide Web Conference (WWW-2002)*, 2002.

[25] D. Dou, D. McDermott, and P. Qi. Ontology translation on the semantic web. In *Proc. Int'l Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE) 2003*, pages 952–969, 2003. LNCS 2888 Springer-Verlag.

[26] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[27] Khun Lee Fung *XSLT: Working with XML and HTML*. Addison-Wesley 2001

[28] M. R. Genesereth, A. M. Keller, and O. M. Duschka. Infomaster: an information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 539–542, 1997.

[29] T. Gruber. Ontolingua: A Translation Approach to Providing Portable Ontology Specifications. *Knowledge Acquisition* , 5(2):199–200, 1993.

[30] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.*, NOV 2004.

[31] A. Y. Halevy. Answering queries using views: A survey. *VLDB J* , 10(4):270–294, 2001.

[32] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. *Lecture Notes in Computer Science*, 1397:27, 1998.

[33] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Halevy. Representing and Reasoning about Mappings between Domain Models. In *Proc. AAAI 2002*, 2002.

[34] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. 27th Int. Conf. on Very Large Data Bases (VLDB).*

[35] A. Maedche, B. Motik, N. Silva, and R. Volz. , mafra - a mapping framework for distributed ontologies. In *Proceedings of EKAW2002*, pages 235–250, 2002.

[36] D. McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003) Available at `ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`.

[37] D. McDermott, M. Burstein, and D. Smith. Overcoming ontology mismatches in transactions with self-describing service agents. In *Proc. Semantic Web Working Symposium*, pages 285–302, 2001.

[38] D. McDermott and D. Dou. Representing Disjunction and Quantifiers in RDF. In *Proceedings of International semantic Web Conference 2002*, pages 250–263, 2002.

[39] D. McGuinness, R.Fikes, J.Rice, and S.Wilder. An Environment for Merging and Testing Large Ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000.

[40] N. F. Noy and M. A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. Technical Report 2000-0831, *Proc. AAAI* **17** Also available as Stanford SMI, 2000.

[41] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd edition).* Prentice Hall, 2002.

[42] L. Serafini, P. Bouquet, B. Magnini, and S. Zanobini. An algorithm for matching contextualized schemas via sat. In *Proc. CONTEX'03*, 2003.

[43] G. Stumme and A. Maedche. Ontology merging for federated ontologies on the semantic web. In *Proceedings of the International Workshop for Foundations of Models for Information Integration (FMII-2001)*, September 2001.

[44] Jeffrey D. Ullman *Principles of Database and Knowledge-Base Systems,* **1**. New York: Computer Science Press. 1988a

[45] Jeffrey D. Ullman *Principles of Database and Knowledge-Base Systems,* **2** New York: Computer Science Press 1988b

Dejing Dou
Department of Computer and Information Science
120 Deschutes Hall
University of Oregon
Eugene, OR 97403-1202
e-mail: `dou@cs.uoregon.edu`

Drew McDermott
Computer Science Department
Yale University
P.O. Box 208285
New Haven, CT 06520-8285
e-mail: `drew.mcdermott@yale.edu`

Peishen Qi
Computer Science Department
Yale University
P.O. Box 208285
New Haven, CT 06520-8285
e-mail: `peishen.qi@yale.edu`