

Modularizing OWL Ontologies

Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, Aditya Kalyanpur
University of Maryland at College Park
bernardo@mindlab.umd.edu, bparsia@isr.umd.edu, {evren, aditya}@cs.umd.edu

ABSTRACT

Modularity in ontologies is key both for large scale ontology development and for distributed ontology reuse on the Web. In this paper, we address the problem of determining and retrieving the fragment of an ontology that captures the essential meaning of a given entity in the ontology. We present an algorithm for extracting fragments that preserve a set of key entailments associated with that entity. Finally, we discuss our implementation integrated in the ontology editor SWOOP and present some promising empirical results. To the best of our knowledge, the method presented in this paper is the first approach to the problem of identifying and retrieving relevant fragments of OWL-DL ontologies that is formally grounded on the semantics of OWL.

1. MOTIVATION

A popular claim about Web ontologies is that they facilitate knowledge reuse and sharing across perspectives and trust boundaries in the open, distributed, decentralized context of the Web. Such a claim precludes large monolithic ontologies. For example, if one wishes to reuse a term from an ontology that one does not control or even fully understand, it is natural to aim for the least possible commitment to the less trusted context. If one agrees with the characterization of one term in a foreign ontology, but not with the rest of the ontology, it would be nice to be able to just reuse the trusted term. Web ontologies should permit multiple parties to use that term alone without explicit, prior agreement.

An effective way to achieve some sort of reuse of the *meaning* of a term, say a class, would be to manually “cut and paste” from the remote ontology certain ax-

ioms considered to be relevant for the characterization of the class. However, in an expressive DL, such as *SHOIN(D)*, it is unfeasible for users to determine, even with the assistance of a reasoner, which axioms should be extracted from the remote ontology and they have little hope of verifying the correctness of the extraction.

The Web Ontology Language does provide the *owl:imports* construct, which allows to include by reference *all* the axioms contained in (and imported by) the remote ontology. Evidently, importing the remote ontology is a way to correctly capture the meaning of the term to be reused. However, some unnecessary information will also be brought, which can be especially problematic in the case of large ontologies.

In general, the ability to identify relevant fragments of ontologies is important for virtually every Semantic Web application. However, even defining what makes a set of axioms a relevant subset of an ontology for a certain task is a controversial issue. In this paper, we provide an algorithm for identifying and extracting relevant fragments of ontologies and justify their relevance by showing that they preserve a certain set of key entailments associated with the entity under consideration.

2. RELATED WORK AND CONTRIBUTIONS

The problem of identifying relevant fragments of an ontology for modularity purposes has been recently addressed in [8] and [6]. In [8], the output is presented as a graph visualization of the different kinds of information contained in the input ontology. However, we identify two main drawbacks in such approach: first, no correspondence between the nodes of the graph and sets of axioms is provided; second, the heuristics considered to generate the visualization only consider a small fragment of OWL that roughly corresponds to RDF-Schema.

The PromptFactor tool [6] uses structural tracing for extracting relevant fragments of ontologies. Although the output in this case is a set of axioms, a formal characterization of the relationship between the semantics of a given term in the input ontology and in the output

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

K-CAP'05, October 2–5, 2005, Banff, Alberta, Canada.
Copyright 2005 ACM 1-59593-163-5/05/0010 ...\$5.00

$$\mathcal{O} = \{ \text{CarOwner} \sqsubseteq \text{Person} \sqcap \exists \text{owns.Car}, \\ \text{Car} \sqsubseteq \text{Vehicle}, \\ \text{Book} \sqsubseteq \text{Publication}, \\ \text{CarOwner} \sqsubseteq \text{Person} \}$$

Figure 1: Example Ontology

fragment is lacking and hence no notion of correctness of the process is established.

Finally, [5] explores partitioning FOL theories to improve theorem prover performance. Our goal in this paper has been very different, since we have examined partitioning for modeling purposes.

3. MODULARITY

In this section, we introduce the intuitive notion of *module* for an entity in the context of a given ontology.

Along the paper, we will use the following notation: \mathcal{O} will denote the ontology under consideration. Capital Greek letters will be used for naming fragments of \mathcal{O} . We will use the term *entity* to refer generically to atomic concepts, atomic roles and individuals. The set of entities that are referenced in a fragment constitutes its *vocabulary*. We will use a triple $V = (V_C, V_R, V_I)$ to represent the vocabulary of the ontology \mathcal{O} , where V_C, V_R, V_I are the sets of atomic concepts, atomic roles and individuals respectively.

Intuitively, the module for an entity is the *minimal* subset of axioms in the ontology that “capture” its meaning “precisely enough” and hence the minimal set of axioms that are required to understand, process, evolve and re-use the entity.

By “capturing the meaning of an entity precisely enough”, we mean that the module for an entity must preserve a set of *crucial entailments* about the entity and *only* those ones; by *minimality*, we mean that none of its proper subsets preserves such a set of crucial entailments. Therefore, the module for an entity in an ontology should be verify two main properties: preservation of certain key entailments and minimality.

In order to illustrate such an intuition, let us consider the simple ontology in Figure 1.

Suppose we want to obtain the module Γ_{Book} corresponding to the concept *Book* in \mathcal{O} . Intuitively, the statement $(\text{Book} \sqsubseteq \text{Publication})$ is the only axiom needed for understanding *Book*, since there is no other statement that constrains the possible models of *Book* in \mathcal{O} . Therefore, $\Gamma_{Book} = \{(\text{Book} \sqsubseteq \text{Publication})\}$. However, the fact of extracting such a fragment rules out certain entailments that held in the original ontology. For ex-

ample, in \mathcal{O} it is entailed that every *Book* is a *Publication* or a *Vehicle*, i.e. $\mathcal{O} \models \{(\text{Book} \sqsubseteq \text{Publication} \sqcup \text{Vehicle})\}$. Such an entailment should intuitively be pruned in a module that captures the essential meaning of *Book* in the ontology. From this example, we observe that, if we enforced the preservation in Γ_{Book} of *all* the entailments involving *Book* in \mathcal{O} , then the module for *Book* would be just \mathcal{O} , i.e. $\Gamma_{Book} = \mathcal{O}$.

The problem then boils down to distinguishing relevant entailments, which should be preserved, from irrelevant entailments, which should be pruned. In Section 4, Theorem 2 we will provide a formal characterization of the minimum set of entailments that must be preserved.

Let us assume now that we want to retrieve the module for *CarOwner*. Intuitively, the module $\Gamma_{CarOwner}$ should entail that a *CarOwner* is a *Person* who owns a *Car* and hence also a *Vehicle*. Therefore, the first candidate for a module for *CarOwner* would be:

$$\Gamma_{CarOwner} = \{ (\text{CarOwner} \sqsubseteq \text{Person}) \\ (\text{CarOwner} \sqsubseteq \text{Person} \sqcap \exists \text{owns.Car}); \\ (\text{Car} \sqsubseteq \text{Vehicle}); \}$$

Although $\Gamma_{CarOwner}$ captures all the entailments we care about, it is not minimal, since the axiom $\text{CarOwner} \sqsubseteq \text{Person}$ is entailed by the rest and hence is *redundant*. By suppressing the redundant axiom, we would obtain the module for *CarOwner*.

In what follows, we will make this first intuition about modules more precise.

4. THE PARTITIONING ALGORITHM

In this section, we present an algorithm for extracting relevant fragments from the input ontology. We will call the obtained fragments *modules* and compare their properties in terms of the intuitive desiderata for a module, discussed in Section 3.

The proofs of the theorems presented in this section are technically involved and lengthy. We refer the interested reader to [1] and [2] for a detailed discussion. Here, we just provide a description of the procedure and some examples, without going into the most formal details.

The main idea of the algorithm is to generate from the input ontology \mathcal{O} a directed labeled graph $G(\mathcal{O})$, called the *partitioning graph*, and then use the graph to find the module for each entity in \mathcal{O} .

The nodes of $G(\mathcal{O})$ are labeled with fragments of \mathcal{O} such that the labels of two different nodes are disjoint ¹ and

¹i.e. do not share any axiom

$$\mathcal{O} = \{ \text{Student} \sqsubseteq \exists \text{enrolledIn.Course}, \\ \text{Student} \sqsubseteq \text{Person}, \\ \text{Professor} \sqsubseteq \exists \text{teaches.Course} \sqcap \exists \text{memberOf.Dept} \\ \text{Paper} \sqsubseteq \text{Publication} \\ \text{Dept.} \sqsubseteq \exists \text{memberOf}^{\neg} . \text{Student} \\ \exists \text{enrolledIn} . \top \sqsubseteq \text{Person} \}$$

Figure 2: Example Ontology

the union of the labels of all the nodes in the graph is precisely $G(\mathcal{O})$ ².

The edges of $G(\mathcal{O})$ are labeled with sets of roles and the labels of two different edges are disjoint. Intuitively, a directed edge (v_i, v_j) between the nodes v_i and v_j means that any change in the axioms contained in the label of v_j may affect the possible models in \mathcal{O} of the axioms contained in the label of v_i . Therefore the partitioning graph provides crucial information about the semantic dependencies between different fragments of the input ontology.

It is worth pointing out here that there exists a very close correspondence between a partitioning graph and a knowledge base in the language of an \mathcal{E} -Connection. Such a correspondence has motivated our algorithmic decisions and has been often exploited in the proofs. Since the introduction of \mathcal{E} -Connections involves a rather sophisticated formal machinery, we will assume along the paper that the reader is only familiar with OWL and refer to [4] and [3] for more details on \mathcal{E} -Connections.

The algorithm consists of three main steps: first, a “safety” check, which is key for ensuring that the obtained fragments preserve certain crucial entailments; second, the generation of the partitioning graph $G(\mathcal{O})$ and, finally, the extraction of the module for each entity from $G(\mathcal{O})$. We next explain in detail each step of the algorithm using as an example the ontology in Figure 2.

1) Safety Check: The presence of certain General Concept Inclusion Axioms (GCIs) may impose general semantic constraints on the ontology *as a whole*. Such constraints can enforce the module for every entity to be the input ontology itself. In order to detect the presence of “dangerous” GCIs, we introduce the notion of safety:

DEFINITION 1. (*safety*)

Let: $g : C \in \mathcal{O} \rightarrow \{T, F\}$ be a function mapping every

²That is why we call such a graph *partitioning graph*, since the axioms of \mathcal{O} are strictly partitioned among the labels of the different nodes

$$\mathcal{O} = \{ \top \sqsubseteq \{a\}, \\ \{a\} \sqsubseteq \exists R.C \}, \\ \Gamma_1 = \{ \top \sqsubseteq \{a\}, \\ \{a\} \sqsubseteq \exists R.C \} \\ \Gamma_2 = \{ C \sqsubseteq \top \}$$

Figure 3: An Unsafe Ontology

concept $C \in \mathcal{O}$ to a boolean value and recursively defined as follows:

- If C is \top , then $g(C) = F$
- Let $C \in V_C$, then $g(C) = T$
- Let C be $\{a\}$, for $a \in V_I$, then $g(C) = T$
- Let C be $D \sqcap E$. If $g(D)=F$ and $g(E)=F$, then $g(C)=F$. Otherwise, $g(C) = T$
- Let C be $D \sqcup E$. If $g(D) = T$ and $g(E) = T$, then $g(C) = T$. Otherwise, $g(C) = F$
- Let C be $\neg D$. If $g(D) = T$, then $g(C) = F$ and if $g(D) = F$, then $g(C) = T$.
- Let C be $\exists P.D$ or $\geq nP$, then $g(C) = T$
- Let C be $\forall P.D$ or $\leq nP$, then $g(C) = F$

The ontology \mathcal{O} is **safe** iff it contains no axiom of the form $C \sqsubseteq D$ s.t. $g(C) = F$ and $g(D) = T$;

It is not hard to verify, using the definition, that the ontology in Figure 2 is safe. In order to understand the potential effect of “dangerous” GCIs, let us consider the the ontology \mathcal{O} in Figure 3, which is *not* safe. Using our algorithm on this ontology (see steps 2 and 3), we would obtain the fragments Γ_1 , for a, R and Γ_2 , for C . However, Γ_2 is not a suitable module for C , since \mathcal{O} entails $C(a)$ and Γ_2 does not, which is certainly a relevant entailment that we should aim at preserving. In such a case, the algorithm determines that the module for every entity is the input ontology itself.

Note that the definition of safety we have provided is *structural* and hence the fact that an ontology does not contain any unsafe axiom does not mean, in principle, that such an axiom cannot be *entailed*, which would compromise our approach.

However, the following theorem shows that unsafe axioms cannot be entailed unless they occur explicitly in the ontology.

```

-Algorithm Partition( $\mathcal{O}$ )
-Input: A SHOIN ontology  $\mathcal{O}$ 
-Output: A partitioning graph  $G(\mathcal{O}) = (\mathbf{V}, \mathbf{E}, \mathcal{L}, \mathcal{V})$ 

 $G(\mathcal{O}) \leftarrow (\{v_0\}, \emptyset, \mathcal{L}, \mathcal{V})$ , with:
 $\mathcal{L}(v_0) = \mathcal{O}$ 
 $\mathcal{V}(X) = v_0$  for each concept or individual  $X$  in  $\mathcal{O}$ 
 $\mathcal{V}(P) = (v_0, v_0)$  for each role  $P$  in  $\mathcal{O}$ 
if  $\mathcal{O}$  not safe, return  $G(\mathcal{O})$ 
for each role  $P$  occurring in  $\mathcal{O}$ ,  $BoundTo(P) \leftarrow \emptyset$ 
Repeat
 $G(\mathcal{O}) \leftarrow DoPartitioningStep(G(\mathcal{O}))$ 
until  $\mathcal{L}(v_0) = \emptyset$ 
 $\mathbf{V} \leftarrow \mathbf{V} - v_0$ 
return  $G(\mathcal{O})$ 

```

Figure 4: Partitioning Algorithm

THEOREM 1. *Let \mathcal{O} with vocabulary V and consistent be safe, then there are no *SHOIN* concepts C, D in the vocabulary V s.t. $g(C) = F$, $g(D) = T$ and $\mathcal{O} \models (C \sqsubseteq D)$.*

2) Generation of the partitioning graph In case of a positive result in the safety check, the algorithm generates in this step a partitioning graph $G(\mathcal{O})$.

In general (see [5], for example), $\{\Gamma_i\}_{1 \leq i \leq n}$ is a **partitioning** of a logical theory Γ if $\Gamma = \bigcup_i \Gamma_i$. Each individual Γ_i is called a **partition**; V_{Γ_i} is its vocabulary (the set of non-logical symbols). In our case, we will consider a *SHOIN* ontology \mathcal{O} with vocabulary $V = (V_C, V_R, V_I)$ and obtain a partitioning of it. The partitioning is defined by means of a labeled directed graph $G(\mathcal{O}) = (\mathbf{V}, \mathbf{E}, \mathcal{L}, \mathcal{V})$, where each node $v_i \in \mathbf{V}$ is labeled with a non-empty partition $\mathcal{L}(v_i) \subseteq \mathcal{O}$, such that $\bigcup_{v \in \mathbf{V}} \mathcal{L}(v) = \mathcal{O}$. In addition we require the partitions to be *disjoint* ($\mathcal{L}(v_i) \cap \mathcal{L}(v_j) = \emptyset$ for $i \neq j$). Each edge $e = (v_i, v_j)$ is labeled with a non-empty set of roles $\mathcal{L}(e)$ occurring in \mathcal{O} and the labels of different edges are also disjoint ($\mathcal{L}(e) \cap \mathcal{L}(e') = \emptyset$ for $e \neq e'$).

In general, given two partitions, their respective vocabularies may *intersect*. In order to define the conditions under which two partitions may share a symbol, we introduce a mapping \mathcal{V} in the graph that assigns to each concept C and individual a occurring in \mathcal{O} a *single* node $\mathcal{V}(C) \in \mathbf{V}$ (respectively $\mathcal{V}(a) \in \mathbf{V}$), and to each role R a single (directed) edge $(\mathcal{V}(R) \in \mathbf{E})$.

Since each entity is mapped through \mathcal{V} into a single node or edge, the function \mathcal{V} allows to “disambiguate” the shared symbols. This mapping will reveal key for determining which axioms from the original ontology will be grouped together in the same partition as well as for retrieving the module for each entity from the partitioning graph.

In what follows we describe the *partitioning algorithm* that generates the partitioning graph.

```

-Algorithm DoPartitioningStep( $G(\mathcal{O})$ )
-Input: A partitioning graph  $G(\mathcal{O})$ 
-Output: Updated graph  $G(\mathcal{O})$ 

Create new node  $v$  with  $\mathcal{L}(v) = \emptyset$  and do  $\mathbf{V} \leftarrow \mathbf{V} \cup v$ 
Select non-deterministically a concept or individual  $X$  in  $\mathcal{O}$ 
with  $\mathcal{V}(X) = v_0$ , or a role  $X$  with  $\mathcal{V}(X) = (v_0, v_0)$ 
if  $X$  a concept or individual, then  $\mathcal{V}(X) \leftarrow v$ 
if  $X$  a role then  $\mathcal{V}(X) \leftarrow (v, v_0)$ 
 $G(\mathcal{O}) \leftarrow MoveEntities(G(\mathcal{O}), v)$ 
 $G(\mathcal{O}) \leftarrow MoveAxioms(G(\mathcal{O}), v)$ 
return ( $G(\mathcal{O})$ )

```

Figure 5: Partitioning Steps

The algorithm performs a succession of *partitioning steps*, as shown in Figure 4. Each step involves a *pair* of nodes in the graph: the node v_0 , called the *source* node, which initially contains in its label the input ontology and from which entities and axioms are removed, and a the node v , the *target* node, generated from scratch, to which these are added. Note that the source node is always v_0 and the target node is different in each step.

At the beginning of each partitioning step (see Figure 5), the algorithm selects non-deterministically an entity X in the source node v_0 and changes the value of $\mathcal{V}(X)$. In the case of a class, for example, $\mathcal{V}(X)$ is updated to v_i , which intuitively means that the class is “moved” to the target node.

This initial change will trigger new ones, according to Figure 7.

The changes in the \mathcal{V} function described in Figure 7 have been devised according to the following principles: First, the modules to be obtained from the partitioning graph *must* preserve the crucial entailments given by Theorem 2, at the end of this section; Second, the nodes of the partitioning graph must contain as fewer axioms as possible, in order to obtain small modules; Finally, the identification and extraction of modules must be computed efficiently.

Depending on the final value of the \mathcal{V} function, some of the axioms in $\mathcal{L}(v_0)$ are removed from $\mathcal{L}(v_0)$ and added to $\mathcal{L}(v)$ and the labels of the edges involving the target and the source nodes are updated.

In Figure 6, we provide the content of the partitioning graph at the end of each partitioning step for the example ontology in Figure 2. We also provide the initial change in the \mathcal{V} function for each step. The reader should be able to reproduce these results using the algorithms in Figures 4, 7 and 8. As a remark, the set $BoundTo(P)$ represents the set of entities that are “forced” to end up in the same node due to the fact that a role P cannot appear in the label of two different edges.

Step1: Initial transition: $\mathcal{V}(Course) \leftarrow v_1$

$\mathbf{V} = \{v_0, v_1\}$ $\mathbf{E} = \{(v_0, v_0), (v_0, v_1)\}$
 $\mathcal{L}(v_0) = \{ Student \sqsubseteq \exists enrolledIn.Course, Student \sqsubseteq Person, Professor \sqsubseteq \exists teaches.Course \sqcap \exists memberOf.Dept, Paper \sqsubseteq Publication, Dept. \sqsubseteq \exists memberOf^-.Student, \exists enrolledIn.\top \sqsubseteq Person \}$
 $\mathcal{L}(v_1) = \{ Course \sqsubseteq \top \}$
 $\mathcal{L}((v_0, v_1)) = \{ enrolledIn, teaches \}$
 $\mathcal{L}((v_0, v_0)) = \{ memberOf, memberOf^- \}$

Step2: Initial transition: $\mathcal{V}(Paper) \leftarrow v_2$

$\mathbf{V} = \{v_0, v_1, v_2\}$ $\mathbf{E} = \{(v_0, v_1), (v_0, v_2)\}$
 $\mathcal{L}(v_0) = \{ Student \sqsubseteq \exists enrolledIn.Course, Student \sqsubseteq Person, Professor \sqsubseteq \exists teaches.Course \sqcap \exists memberOf.Dept, Dept. \sqsubseteq \exists memberOf^-.Student, \exists enrolledIn.\top \sqsubseteq Person \}$
 $\mathcal{L}(v_1) = \{ Course \sqsubseteq \top \}$
 $\mathcal{L}(v_2) = \{ Paper \sqsubseteq Publication \}$
 $\mathcal{L}((v_0, v_1)) = \{ enrolledIn, teaches \}$
 $\mathcal{L}((v_0, v_0)) = \{ memberOf, memberOf^- \}$

Step3: Initial transition: $\mathcal{V}(Dept) \leftarrow v_3$

$\mathbf{V} = \{v_0, v_1, v_2, v_3\}$
 $\mathbf{E} = \{(v_0, v_1), (v_0, v_3), (v_3, v_0)\}$
 $\mathcal{L}(v_0) = \{ Student \sqsubseteq \exists enrolledIn.Course, Student \sqsubseteq Person, Professor \sqsubseteq \exists teaches.Course \sqcap \exists memberOf.Dept, \exists enrolledIn.\top \sqsubseteq Person \}$
 $\mathcal{L}(v_1) = \{ Course \sqsubseteq \top \}$
 $\mathcal{L}(v_2) = \{ Paper \sqsubseteq Publication \}$
 $\mathcal{L}(v_3) = \{ Dept. \sqsubseteq \exists memberOf^-.Student \}$
 $\mathcal{L}((v_0, v_1)) = \{ enrolledIn, teaches \}$
 $\mathcal{L}((v_0, v_3)) = \{ memberOf \}$
 $\mathcal{L}((v_3, v_0)) = \{ memberOf^- \}$

Step4: Initial transition: $\mathcal{V}(Student) \leftarrow v_4$

$\mathbf{V} = \{v_0, v_1, v_2, v_3, v_4\}$
 $\mathbf{E} = \{(v_4, v_1), (v_4, v_3), (v_3, v_4)\}$
 $\mathcal{L}(v_0) = \emptyset$
 $\mathcal{L}(v_1) = \{ Course \sqsubseteq \top \}$
 $\mathcal{L}(v_2) = \{ Paper \sqsubseteq Publication \}$
 $\mathcal{L}(v_3) = \{ Dept. \sqsubseteq \exists memberOf^-.Student \}$
 $\mathcal{L}(v_4) = \{ Student \sqsubseteq \exists enrolledIn.Course, Student \sqsubseteq Person, Professor \sqsubseteq \exists teaches.Course \sqcap \exists memberOf.Dept, \exists enrolledIn.\top \sqsubseteq Person \}$
 $\mathcal{L}((v_4, v_1)) = \{ enrolledIn, teaches \}$
 $\mathcal{L}((v_4, v_3)) = \{ memberOf \}$
 $\mathcal{L}((v_3, v_4)) = \{ memberOf^- \}$

Figure 6: A Decomposition into a Partitioning Graph

-Algorithm MoveEntities($G(\mathcal{O}), v$)
-Input: A partitioning graph $G(\mathcal{O}) = (\mathbf{V}, \mathbf{E}, \mathcal{L}, \mathcal{V})$
The target node v in the current partitioning step
-Output: A partitioning graph with updated mapping \mathcal{V}

Repeat
for all concept $C \neq \top$ with $\mathcal{V}(C) = v_0$
if any of the following conditions holds:
1) $(C \sqsubseteq D)$ or $(D \sqsubseteq C) \in \mathcal{L}(v_0)$, and $\mathcal{V}(D) = v$
2) $C(a) \in \mathcal{L}(v_0)$ and $\mathcal{V}(a) = v$
3) $\exists P.C$ or $\forall P.C \in \mathcal{L}(v_0)$ and $\mathcal{V}(P) \in \{(v_0, v), (v, v)\}$
4) $(C \sqcap D)$ or $(C \sqcup D) \in \mathcal{L}(v_0)$ and $\mathcal{V}(D) = v$
5) C of the form $D \sqcap E$, or $D \sqcup E$ and $\mathcal{V}(E) = v$ or $\mathcal{V}(D) = v$
6) C a restriction $\exists P.D, \forall P.D, \geq nP$ or $\leq nP$, and $\mathcal{V}(P) \in \{(v, v_0), (v, v), (v_j, v_0)\}$
7) C of the form $\{a\}$ and $\mathcal{V}(a) = v$
8) $(\neg C) \in \mathcal{L}(v_0)$ and $\mathcal{V}(\neg C) = v$
9) $C, E \in BoundTo(P)$ and $\mathcal{V}(E) = v$
then $\mathcal{V}(C) \leftarrow v$
if 3) has held, **then** $BoundTo(P) \leftarrow BoundTo(P) \cup \{a\}$
for all individual a with $\mathcal{V}(a) = v_0$
if any of the following conditions holds:
1) $C(a) \in \mathcal{L}(v_0)$ and $\mathcal{V}(C) = v$
2) $\{a\} \in \mathcal{L}(v_0)$ and $\mathcal{V}(\{a\}) = v$
3) $P(a, b) \in \mathcal{L}(v_0)$ and $\mathcal{V}(P) \in \{(v, v_0), (v, v), (v_j, v_0)\}$
4) $P(b, a) \in \mathcal{L}(v_0)$ and $\mathcal{V}(P) \in \{(v_0, v), (v, v)\}$
then $\mathcal{V}(a) \leftarrow v$
if 4) has held, **then** $BoundTo(P) \leftarrow BoundTo(P) \cup \{a\}$
for all role P with $\mathcal{V}(P) \in \{(v_0, v_0), (v_0, v), (v, v_0), (v_0, v_j)\}$
if $(P \sqsubseteq Q)$ or $(Q \sqsubseteq P) \in \mathcal{L}(v_0)$, and $\mathcal{V}(Q) \in \{(v_0, v), (v, v_0), (v, v), (v_j, v_0)\}$ **then** $\mathcal{V}(P) \leftarrow \mathcal{V}(Q)$
if $\mathcal{V}(P) \in \{(v_0, v), (v, v_0)\}$ and P transitive, **then** $\mathcal{V}(P) \leftarrow (v, v)$
if $P(a, b) \in \mathcal{L}(v_0)$, $\mathcal{V}(a) = v$, and $\mathcal{V}(P) = (v_0, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v, v_0)$; add a to $BoundTo(P)$ and $\mathcal{V}(P) = (v_0, v)$, **then** $\mathcal{V}(P) \leftarrow (v, v)$; remove a from $BoundTo(P)$ and $\mathcal{V}(P) = (v_0, v_j)$, **then** $\mathcal{V}(P) \leftarrow (v_j, v_0)$
if $P(a, b) \in \mathcal{L}(v_0)$ and $\mathcal{V}(b) = v$, and $\mathcal{V}(P) = (v_0, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v_0, v)$; add b to $BoundTo(P)$ and $\mathcal{V}(P) = (v, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v, v)$; remove b from $BoundTo(P)$
if D a restriction on P , $\mathcal{V}(D) = (v_0, v)$ and $\mathcal{V}(P) = (v_0, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v, v_0)$ and $\mathcal{V}(P) = (v_0, v)$, **then** $\mathcal{V}(P) \leftarrow (v, v)$ and $\mathcal{V}(P) = (v_0, v_j)$, **then** $\mathcal{V}(P) \leftarrow (v_j, v_0)$
if $\exists P.C$ or $\forall P.C \in \mathcal{L}(v_0)$, $\mathcal{V}(C) = v$ and $\mathcal{V}(P) = (v_0, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v_0, v)$ and $\mathcal{V}(P) = (v, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v, v)$
if $\mathcal{V}(P) = (v_0, v_0)$ and $\mathcal{V}(Inv(P)) = (v_0, v)$, **then** $\mathcal{V}(P) \leftarrow (v, v_0)$
if $\mathcal{V}(P) = (v_0, v_0)$ and $\mathcal{V}(Inv(P)) = (v, v_0)$, **then** $\mathcal{V}(P) \leftarrow (v_0, v)$
if $\mathcal{V}(P) \in \{(v_0, v_0), (v_0, v), (v, v_0)\}$ and $\mathcal{V}(Inv(P)) = (v, v)$, **then** $\mathcal{V}(P) \leftarrow (v, v)$
until no change in \mathcal{V} is triggered
return $G(\mathcal{O})$

Figure 7: Moving Entities

Finally, although the initial change in each partitioning step is chosen non-deterministically, it is possible to prove that the result is deterministic and, given an input ontology, the same partitioning graph will always be obtained.

3) Generation of the modules: The module for each entity is obtained from the partitioning graph using the algorithm in Figure 9. According to the Figure, if $\mathcal{V}(X) = v_i$, the **module** for X in \mathcal{O} is the union of all the axioms contained in the nodes that are accessible from v_i through a directed path in $G(\mathcal{O})$. There are cases, however, where the module computed that way does not contain all the required axioms for satisfying Theorem 2. For example, consider the following ontology:

```

-Algorithm MoveAxioms( $G(\mathcal{O}), v$ )
-Input: A partitioning graph  $G(\mathcal{O})$ 
The target node  $v$  in the current partitioning step
-Output: An updated partitioning graph  $G(\mathcal{O})$ 

for each Axiom  $A \in \mathcal{L}(v_0)$ 
if  $A$  is of any of the following forms:
1)  $C \sqsubseteq D$  and  $\mathcal{V}(C) = \mathcal{V}(D) = v$ 
2)  $C(a)$  and  $\mathcal{V}(a) = \mathcal{V}(C) = v$ 
3)  $P \sqsubseteq Q$ , and  $\mathcal{V}(P) = \mathcal{V}(Q)$ , with  $\mathcal{V}(P) \in \{(v, v_0), (v, v), (v_j, v_0)\}$ 
4)  $Trans(P)$  and  $\mathcal{V}(P) \in \{(v, v_0), (v, v)\}$ 
5)  $P(a, b)$  and  $\mathcal{V}(P) \in \{(v, v_0), (v, v), (v_j, v_0)\}$ 
then  $\mathcal{L}(v_0) \leftarrow \mathcal{L}(v_0) - \{A\}$ 
and  $\mathcal{L}(v) \leftarrow \mathcal{L}(v) \cup \{A\}$ 
for each  $P$ , s.t.  $\mathcal{V}(P) = (v_0, v)$ , do
 $L((v_0, v)) \leftarrow L((v_0, v)) \cup \{P\}$  for each  $P \in \mathcal{L}((v_j, v_0))$  with  $v_j \neq v$ 
if  $\forall X \in BoundTo(P), \mathcal{V}(X) = v$ , then
 $\mathcal{L}((v_j, v_0)) = \mathcal{L}((v_j, v_0)) - \{P\}$ 
if  $\mathcal{L}((v_j, v_0)) = \emptyset$  remove edge  $(v_j, v_0)$  from  $G(\mathcal{O})$ 
if  $(v_j, v) \notin E$ , add edge  $(v_j, v)$  to  $G(\mathcal{O})$ 
 $\mathcal{L}((v_j, v)) = \mathcal{L}((v_j, v)) \cup \{P\}$ 
return  $G(\mathcal{O})$ 

```

Figure 8: Moving Axioms

```

-Algorithm GenerateModule( $G(\mathcal{O}), X$ )
-Input: The partition graph  $G(\mathcal{O})$ 
An entity  $X$  in  $\mathcal{O}$ 
-Output: The module  $\Gamma$  for  $X$  in  $\mathcal{O}$ 

 $v \leftarrow \mathcal{V}(X)$ 
 $\Gamma \leftarrow \mathcal{L}(v)$ 
Add to  $\Gamma$  all axioms in the label of the nodes accessible from  $v$ .
if  $\mathcal{L}(v)$  has nominals, then
for each predecessor  $w$  of  $v$  in  $G(\mathcal{O})$ 
Select any entity  $Z$  in  $\mathcal{L}(w)$ 
 $\Gamma \leftarrow \Gamma \cup GenerateModule(G(\mathcal{O}), Z)$ 
return  $\Gamma$ 
for each predecessor  $w$  of  $v$  s.t.  $P(a, b) \in \mathcal{L}(w)$  with  $P \in \mathcal{L}((w, v))$ 
Select any entity  $Z$  in  $\mathcal{L}(w)$ 
 $\Gamma \leftarrow \Gamma \cup GenerateModule(G(\mathcal{O}), Z)$ 
return  $\Gamma$ 

```

Figure 9: Generation of Modules

$$\mathcal{O} = \{C \sqsubseteq \forall R.B ; B \sqsubseteq E \\ C(a) ; R(a, b)\}$$

The partitioning algorithm would generate a graph with two nodes v, w , with $\mathcal{L}(v) = \{C \sqsubseteq \forall R.B; C(a); R(a, b)\}$ and $\mathcal{L}(w) = \{B \sqsubseteq E\}$ connected by an edge (v, w) with $\mathcal{L}((v, w)) = \{R\}$. The module Γ for B would be just $\Gamma = \mathcal{L}(w)$; however $\mathcal{O} \models B(b)$, which is not entailed in Γ , thus violating Theorem 2. The problem is caused by the presence of the axiom $R(a, b)$, where R is in the label of the edge connecting v and w . In these cases (see Figure 9) we need to “backtrack” in the graph in order to satisfy the theorem, and we would add the axioms in $\mathcal{L}(v)$ to the module for B . Similar effects could occur in case w contained nominals (i.e. enumerations of objects).

By construction, an entity cannot have two different modules and, given two entities, their respective modules coincide iff they correspond to the same node in $G(\mathcal{O})$.

The fundamental advantage of our approach is charac-

terized by the following result:

THEOREM 2. *The module $\Gamma \sqsubseteq \mathcal{O}$ for an atomic concept $A \in V_C$ in \mathcal{O} verifies the following:*

1. $\Gamma \models (A \sqsubseteq B) \Leftrightarrow \mathcal{O} \models (A \sqsubseteq B)$ for every $B \in V_C$
2. $\Gamma \models (B \sqsubseteq A) \Leftrightarrow \mathcal{O} \models (B \sqsubseteq A)$ for every $B \in V_C$
3. $\Gamma \models A(a) \Leftrightarrow \mathcal{O} \models A(a)$ for every $a \in V_I$
4. A is satisfiable (unsatisfiable) in Γ iff it is satisfiable (unsatisfiable) in \mathcal{O}

The module Γ for a role $P \in V_R$ in \mathcal{O} verifies the following:

1. $\Gamma \models (P \sqsubseteq Q) \Leftrightarrow \mathcal{O} \models (P \sqsubseteq Q)$ for every $Q \in V_R$
2. $\Gamma \models (Q \sqsubseteq P) \Leftrightarrow \mathcal{O} \models (Q \sqsubseteq P)$ for every $Q \in V_R$
3. $\Gamma \models Domain(P, A) \Leftrightarrow \mathcal{O} \models Domain(P, A)$ for every $A \in V_C$
4. $\Gamma \models Range(P, A) \Leftrightarrow \mathcal{O} \models Range(P, A)$ for every $A \in V_C$
5. $\Gamma \models Transitive(P) \Leftrightarrow \mathcal{O} \models Transitive(P)$
6. $\Gamma \models Functional(P) \Leftrightarrow \mathcal{O} \models Functional(P)$
7. $\Gamma \models P(a, b) \Leftrightarrow \mathcal{O} \models P(a, b)$ for every $a, b \in V_I$

The module Γ for an individual $a \in V_I$ in \mathcal{O} verifies the following:

1. $\Gamma \models A(a) \Leftrightarrow \mathcal{O} \models A(a)$ for every $A \in V_C$
2. $\Gamma \models P(a, b) \Leftrightarrow \mathcal{O} \models P(a, b)$ for every $P \in V_R, b \in V_I$
3. $\Gamma \models P(b, a) \Leftrightarrow \mathcal{O} \models P(b, a)$ for every $P \in V_R, b \in V_I$

Moreover, the modules can be obtained in polynomial time, as shown in the following theorem:

THEOREM 3. *The algorithms described in Figures 4 and 9 are worst-case quadratic in the size of the input ontology.*

Given the way the partitioning graph is constructed, we can ensure that the retrieved modules are the minimal ones we can obtain:

Ontology	Concepts	Roles	Indiv.	Modules	Time(s)
OWL-S	51	54	9	17	0.29
NASA	1537	102	194	43	2.8
GALEN	2749	413	0	2	11
NCI	27652	71	0	17	45

Table 1: Some Partitioned Ontologies

THEOREM 4. *Given an entity, its module is the minimal fragment satisfying the properties in Theorem 2 that can be obtained from the partitioning graph.*

However, the modules we obtain are not the minimal ones verifying Theorem 2. In particular, a node in a partitioning graph may contain redundant axioms (see example in Figure 1).

Finally, note that the partitioning algorithm presented in this section is *completely automatic*. No user intervention is required at any stage of the process.

5. IMPLEMENTATION AND DISCUSSION

We have implemented the partitioning algorithm on top of Manchester’s OWL-API and provided a UI in the ontology editor SWOOP³. We have applied our algorithm to a set of OWL ontologies available on the Web.

Table 1 summarizes the results obtained for the following relevant cases: the OWL-S ontologies about Web Services, the NCI⁴ Thesaurus and GALEN biomedical ontologies and NASA’s SWEET-JPL ontologies on Earth Sciences.

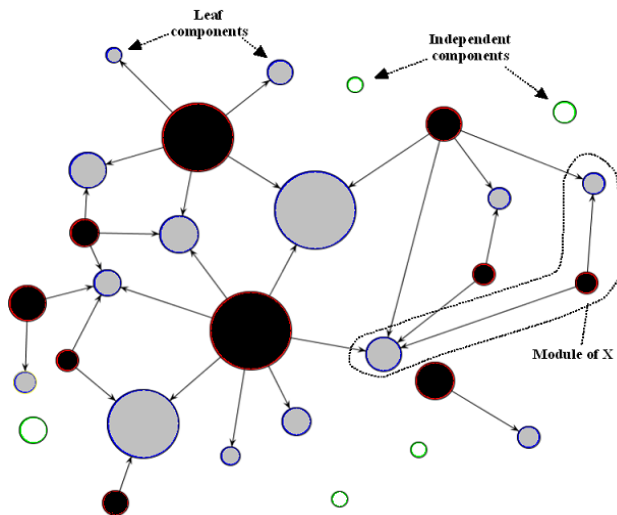


Figure 10: Partitioning Graph for OWL-S

The *OWL-S ontologies* describe Web services and are coupled together using *owl:imports*. We have collapsed

³<http://www.mindswap.org/2004/SWOOP>

⁴National Cancer Institute

the ontologies into a single one respecting the semantics of *owl:imports* and applied the algorithm in Section 4 to the result. The partitioning graph is shown in Figure 10. The ontology exhibits a nice decomposition, since a significant proportion of nodes correspond to independently or leaf nodes (white and grey nodes respectively), which is ideal for re-use. Interestingly, there is an improvement in modularity for every entity, in the sense that every module is *strictly smaller* than the ontology as a whole.

The NCI Thesaurus is a huge, carefully designed, ontology dealing with the biomedical domain. NCI has become a reference terminology covering areas of basic and clinical science, built with the goal of facilitating translational research in cancer. It contains information about diseases, drugs, anatomy, genes, gene products, techniques, and biological processes, among others, all with a cancer-centric focus in content. The decomposition obtained for NCI, shown in Figure 11, exhibits similar nice features as the one for OWL-S.

It is worth emphasizing here that, although the partitioning graph has been used in this paper just as an intermediate representation for obtaining modules, it also has an intrinsic value for modeling. The partitioning graph provides interesting information about the way the ontology has been modeled. First, the axioms in the label of each node represent a well-defined sub-domain within the ontology, intuitively disjoint from the rest. For example, in NCI the knowledge about genes, drugs, medical techniques, etc. are each represented in a different node. These domains are pair-wise disjoint in the sense that they do not share instances (a drug is not a gene and vice-versa). The connections also provide interesting information, since they suggest which domains within the ontology are most relevant. For example, the graph for NCI shows that the node dealing with *genes* is the one that contains the largest number of “outgoing” edges, which implies that genes are central to the ontology. Other nodes, like the one dealing with anatomical structures, are “leaf components” in the graph, and hence represent “secondary” domains.

GALEN is a large medical ontology, represented in the Description Logic *SHF*, designed for supporting clinical information systems. GALEN is conceptually composed of a “Top level” (or upper) ontology, which contains generic (i.e. domain independent) concepts like “Process” or “Substance” and a *Domain Ontology* that contains concepts such as “Gene” or “Research Institution”, which are specific to a certain application domain [7]. One of the goals of this design, according to [7], is to increase modularity by reducing the effort in maintenance and enhancing re-use. However, GALEN cannot be modularized, in the sense modularity is understood in this paper. The main reason is the presence of the upper level ontology, which prevents a good partitioning.

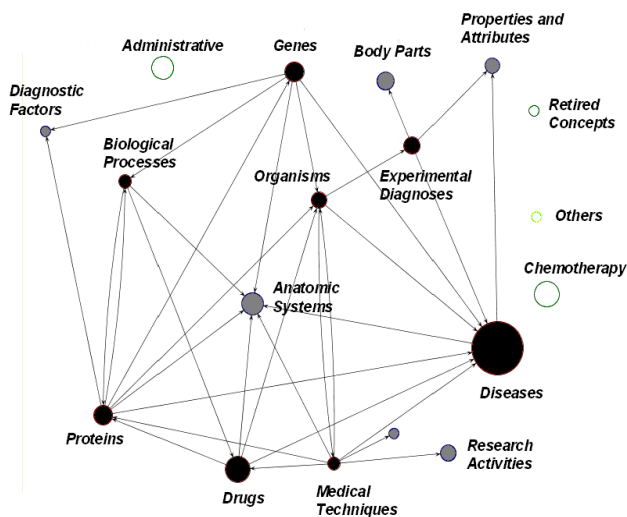


Figure 11: Partitioning Graph for NCI

Note that the preservation of the entailments provided in Theorem 2 immediately enforces the existence of a single node in the graph, since every concept and property are ultimately a subsumed by the top concept and role respectively. The small second partition obtained just shows that there is an unused concept in the ontology.

Finally, the ontologies within the Semantic Web for Earth and Environmental Terminology (SWEET) constitute NASA's effort for providing a formalization of the Earth Science domain. The SWEET ontologies include several thousand terms, spanning a broad extent of Earth Science and related concepts using OWL⁵. As in the case of OWL-S, we have collapsed the SWEET-JPL ontologies into a single one respecting the semantics of *owl:imports* and applied the partitioning algorithm.

According to our results, SWEET-JPL is an example in which the use *owl:imports* to produce an "ad-hoc" modularization of the ontology turns out to be misleading. Our partitioning shows a mismatch between the physically distinct units (the imported ontologies) and the semantically distinct ones. The partitioning also reveals a significant number of small independent nodes. The existence of these small independent chunks of knowledge is hard to detect by direct inspection of the original ontologies⁶ and is not desirable from a modeling perspective, unless one actually wanted to evolve them separately.

⁵The ontologies can be downloaded from <http://sweet.jpl.nasa.gov/sweet>

⁶Actually, the use of *owl:imports* makes it even harder

6. CONCLUSION

In this paper, we have presented a method for automatically identifying and extracting relevant fragments of ontologies, called modules, with precise semantic guarantees. Our method encompasses the full expressive power of OWL-DL and provides a good computational performance. Our initial experimental results with real-world ontologies show that for a significant number of entities the modules we obtain can be notably smaller than the original ontology, which facilitates re-use, processability, understandability and maintenance.

7. REFERENCES

- [1] B. Cuenca-Grau. *Combination and Integration of Ontologies on the Semantic Web*. PhD thesis, Universidad de Valencia, 2005. Submitted.
- [2] B. Cuenca-Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Automatic partitioning of OWL ontologies using \mathcal{E} -connections. In *Proc. of DL 2005*, 2005.
- [3] B. C. Grau, B. Parsia, and E. Sirin. Combining OWL ontologies using \mathcal{E} -connections. 2005. To Appear in *Journal of Web Semantics*.
- [4] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyashev. \mathcal{E} -connections of abstract description systems. *Artificial Intelligence 156(1):1-73*, 2004.
- [5] B. MacCartney, S. A. McIlraith, E. Amir, and T. Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proc. of IJCAI 2003*, 2003.
- [6] N. Noy and M. Musen. The prompt suite: Interactive tools for ontology mapping and merging. *Int. J. of Human-Computer Studies*, 6(59), 2003.
- [7] A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms, including OWL. In *Proc. of the 16th International FLAIRS Conference*. AAAI, 2003.
- [8] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large class hierarchies. In *Proc. of ISWC 2004*, 2004.