Modularizing OWL Ontologies

Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, Aditya Kalyanpur
University of Maryland, College Park
bernardo@mindswap.org, 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 identifying and extracting those fragments and justify their relevance by showing that they preserve a certain set of key entailments associated to the entity under consideration. 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 an 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 minimal set of axioms that "capture" the meaning of that specific class in the remote ontology.

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. Our method provides a formal notion of what it is meant for an axiom of an ontology to be relevant to the meaning of a term, it encompasses the full expressive power of OWL, it is completely automatic and finally it helps identifying modeling errors and omissions.

$$\begin{aligned}\mathcal{O} = \{&CarOwner \sqsubseteq Person \sqcap \exists owns.Car,\\ &Car \sqsubseteq Vehicle,\\ &Book \sqsubseteq Publication,\\ &CarOwner \sqsubseteq Person\ \}\end{aligned}$$

Figure 1: Example Ontology

2 Modularity

By "capturing the meaning of an entity", 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 $(Book \sqsubseteq Publication)$ 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, $\Gamma_{Book} = \{(Book \sqsubseteq Publication)\}$. However, the fact of extracting such a fragment rules out certain entailments that held in the original ontology. For example, in $\mathcal{O}$ it is entailed that every *Book* is a *Publication* or a *Vehicle*, i.e. $\mathcal{O} \models \{(Book \sqsubseteq Publication \sqcup Vehicle\}$. 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 Theorem 1 we provide a formal characterization of the minimum set of entailments that must be preserved.

3 The Partitioning Algorithm

The algorithm consists of three main steps:

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. Safety is required in order to detect the presence of "dangerous" GCIs.

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

The algorithm performs a succession of *partitioning steps*, each of which involves the generation of a new node in the graph. The algorithm non-deterministically adds an axiom of the original ontology to the label of the generated node and then moves all the relevant axioms associated to it. Although the axiom to be moved initially at each step is chosen non-deterministically, we have shown that the result is deterministic, that is, given an input ontology, the same partitioning graph will always be obtained.

3)Generation of *modules* from $G(\mathcal{O})$ The module for an entity X corresponding to a node v_i is the union of all the axioms contained in the nodes that are accessible from v_i through a directed path in $G(\mathcal{O})$.

The fundamental advantage of our approach is characterized by the following result:

Theorem 1 *The module* $\Gamma \subseteq O$ *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* $P \in V_R, b \in V_I$

[1]i.e. do not share any axiom

[2]That 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

Ontology	Atomic 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

Moreover, the modules can be obtained in polynomial time. 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.

We have implemented the partitioning algorithm on top of Manchester's OWL-API and provided a UI in the ontology editor SWOOP[3]. We have applied our algorithm to a set of OWL ontologies available on the Web.

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

All the details about the algorithm, proofs and discussion can be found online in [1].

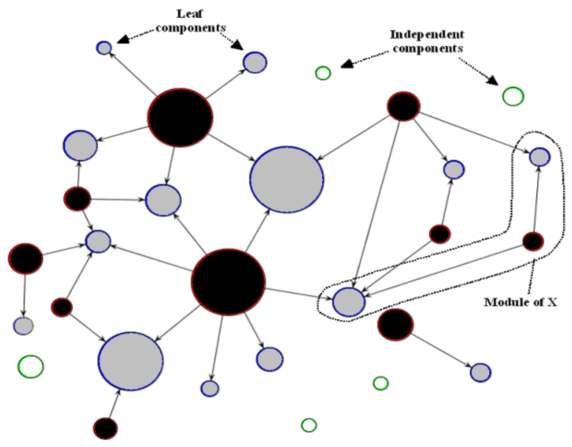


Figure 2: **Partitioning Graph for OWL-S**

References

[1] B. Cuenca-Grau. *Combination and Integration of Ontologies on the Semantic Web.* PhD thesis, Universidad de Valencia, 2005. Available at http://www.mindswap.org/2004/multipleOnt/papers/Dissertation.pdf.

[3]http://www.mindswap.org/2004/SWOOP

Demo description

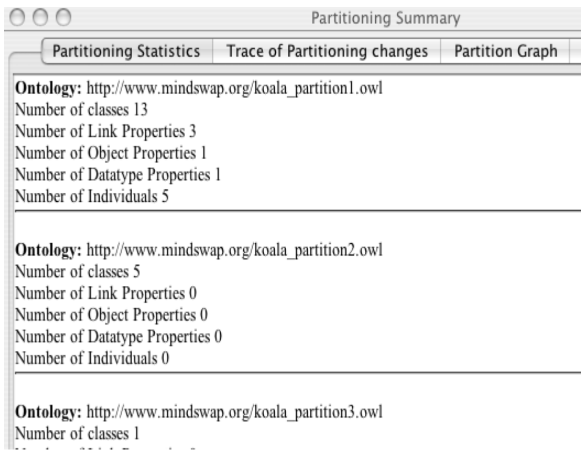
Our implementation of module extraction is hosted and has a UI in our ontology editor Swoop. The demo will run as follows:

We load and generate the partitioning graph for the toy example ontology Koala ¡http://protege.stanford.edu/plugins/owl/owl-library/koala.owl¿. The UI will display an overview of the partitions, a trace of the partitioning process, and a visualization of the partition graph (see figures 3-5).

The user will then be directed to select specific classes or properties to see what the resultant module is like, which they can examine in Swoop (or load into a tool of their choice).

Then we will load a large ontology of the user's choice out of the NCI Thesauras, SWEET JPL, or the OWL-S ontologies (all discussed in the poster) and the above process repeated (to give them a sense of a large ontology).

Then the user may try to modularize an arbitrary ontology of their choice (that is Web accessible — so a net connection would be required) OR we will load the hard-to-partition ontology GALEN and show which aspects of the modeling cause difficulties with our algorithm.

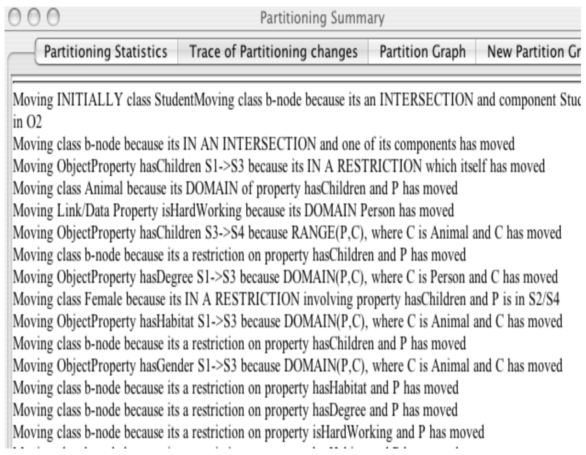


Ontology: http://www.mindswap.org/koala_partition1.owl
Number of classes 13
Number of Link Properties 3
Number of Object Properties 1
Number of Datatype Properties 1
Number of Individuals 5

Ontology: http://www.mindswap.org/koala_partition2.owl
Number of classes 5
Number of Link Properties 0
Number of Object Properties 0
Number of Datatype Properties 0
Number of Individuals 0

Ontology: http://www.mindswap.org/koala_partition3.owl
Number of classes 1

Figure 3: **The overview of the partitions for the Koala ontology including statistics about the component.**



```
Moving INITIALLY class StudentMoving class b-node because its an INTERSECTION and component Stuc
in O2
Moving class b-node because its IN AN INTERSECTION and one of its components has moved
Moving ObjectProperty hasChildren S1->S3 because its IN A RESTRICTION which itself has moved
Moving class Animal because its DOMAIN of property hasChildren and P has moved
Moving Link/Data Property isHardWorking because its DOMAIN Person has moved
Moving ObjectProperty hasChildren S3->S4 because RANGE(P,C), where C is Animal and C has moved
Moving class b-node because its a restriction on property hasChildren and P has moved
Moving ObjectProperty hasDegree S1->S3 because DOMAIN(P,C), where C is Person and C has moved
Moving class Female because its IN A RESTRICTION involving property hasChildren and P is in S2/S4
Moving ObjectProperty hasHabitat S1->S3 because DOMAIN(P,C), where C is Animal and C has moved
Moving class b-node because its a restriction on property hasChildren and P has moved
Moving ObjectProperty hasGender S1->S3 because DOMAIN(P,C), where C is Animal and C has moved
Moving class b-node because its a restriction on property hasHabitat and P has moved
Moving class b-node because its a restriction on property hasDegree and P has moved
Moving class b-node because its a restriction on property isHardWorking and P has moved
```

Figure 4: **A trace of the run, showing why each axiom was moved to its final partition.**

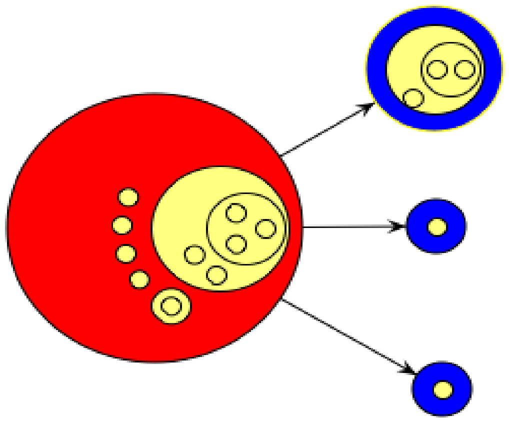


Figure 5: **A visualization of the partition graph for the Koala ontology, including a representation of the class structure of each partition.**