

Modularizing OWL Ontologies

Bernardo Cuenca Grau^{1,2}, Bijan Parsia¹, Evren Sirin¹, and Aditya Kalyanpur¹

¹ Maryland Information and Network Dynamics Laboratory, USA

² Departamento de Informatica, Universidad de Valencia, Spain

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 subset of an ontology that captures the essential meaning of a given entity in the ontology. However, even defining what makes a certain set of axioms a relevant subset of an ontology for a certain task is a controversial issue. In this paper, we provide such a definition by introducing the notion of *semantic encapsulation* of an entity within an ontology. Such a notion will motivate a formal definition of *module*. We then provide an algorithm for finding and retrieving the module that encapsulates the meaning of each entity in a given ontology, an optimized implementation and some promising empirical results.

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.

For example, suppose one wants to annotate a word in a document using a class from an ontology, published elsewhere on the Web. It would be natural for a Semantic Web application to retrieve *only* the minimal set of axioms that “capture” the meaning of that specific class in the remote ontology. However, at the current stage of research and deployment, it is even unclear what it is meant by “capturing” the meaning of an entity within an ontology.

A naive RDF user would tend to believe that the mere use of the URI of the class in his/her document would be sufficient to “import” its meaning. However, in RDF, as well as in OWL, the meaning of a URI is *local* to the document in which it is used. In other words, there is no effective difference between reusing a URI by direct reference and coining a new one.

A more effective way to achieve some sort of reuse of the *meaning* of the class would be to manually “cut and paste” from the remote ontology certain

axioms considered to be relevant for the characterization of the class c . However, in an expressive DL such as $SHOIN(\mathcal{D})$ it becomes almost unfeasible for users to determine, even with the assistance of a reasoner, which axioms should be brought locally as well as which entailments from the original ontology should be preserved in the retrieved set of axioms.

The final alternative is to use the means provided by OWL for explicitly including information from other ontologies. The Web Ontology Language provides 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 class to be reused. However, it is also evident that some unnecessary information will also be brought, which can be specially problematic in the case of large ontologies.

In general, the ability to identify relevant sub-parts of ontologies is important for virtually every Semantic Web application. For example, in ontology engineering, achieving some sort of “modularity” is a key requirement to facilitate collaborative, large scale, long term development. Modularity worthy of the name should provide benefits in the following aspects: processability of the ontology by applications, evolvability and maintenance, knowledge reuse and understandability for humans.

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 such a definition by introducing the notion of *semantic encapsulation* of an entity within an ontology. Such a notion will motivate a formal definition of *module*.

We then address the problem of finding those relevant axioms for each entity in the ontology. The main idea of our approach is to break the original ontology into a specific *combination* of smaller ontologies, called an \mathcal{E} -Connection [6] [2] [4]. Intuitively, each ontology in the resulting combination models a different sub-domain of the input ontology, while the “connections” specify how each ontology in the combination is using information from the others. We have found that the information provided by the \mathcal{E} -Connection is crucial for identifying relevant fragments of the input ontology.

2 Modularity

In this section, we introduce the concept of *module* of an entity in the context of a given ontology. Intuitively, the module of an entity is the minimal subset of axioms in the ontology that “capture” its meaning “precisely enough”. We will use the term *component* of an ontology to denote any subset of its axioms. Our goal will be to formally characterize what it is meant by a component to “capture the meaning of an entity in the 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 components of \mathcal{O} . We will use the term *entity* to refer generically to atomic classes, atomic roles and individuals. The set of entities that are referenced in a component 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 classes, atomic roles and individuals respectively. An additional subscript will be used to denote the vocabulary of a certain component and hence $V_\Gamma = (V_{C_\Gamma}, V_{R_\Gamma}, V_{I_\Gamma})$ represents the vocabulary of the component Γ

We begin by introducing the notion of *semantic encapsulation*.

Definition 1 (*Semantic Encapsulation*)

A component Γ of \mathcal{O} **encapsulates** the atomic class $A \in V_C$ if:

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 in Γ iff it is satisfiable in \mathcal{O}

Γ encapsulates the role $P \in V_R$ iff:

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. $\Gamma \models P(a, b) \leftrightarrow \mathcal{O} \models P(a, b)$ for every $a, b \in V_I$

Γ encapsulates the individual $a \in V_I$ iff:

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 $P \in V_R, b \in V_I$

Intuitively, a component encapsulates an entity in an ontology if it preserves a basic set of entailments. In the case of classes, for example, a component must preserve all the subsumption relations with other classes as well as all the instances of the class. For example, let \mathcal{O} be the following simple ontology:

$$\mathcal{O} = \{(CarOwner \sqsubseteq Person \sqcap \exists owns.Car); (Car \sqsubseteq Vehicle); (Book \sqsubseteq Publication)\}$$

The component $\Gamma_1 = \{(Book \sqsubseteq Publication)\}$ encapsulates the classes *Book* and *Publication*. On the other hand, the component $\Gamma_2 = \{(CarOwner \sqsubseteq Person \sqcap \exists owns.Car)\}$ encapsulates *Person* and *CarOwner* while $\Gamma_3 = \{Car \sqsubseteq Vehicle\}$ encapsulates *Car* and *Vehicle*. Finally, the component that encapsulates the property *owns* is just the empty set.

The fact of encapsulating an entity in a component rules out certain entailments that held in the original ontology. Some of them are irrelevant, but others are not. For example, in \mathcal{O} it is entailed that every *Car* is a *Vehicle* or a *Publication*, i.e. $\mathcal{O} \models \{(Car \sqsubseteq Vehicle \sqcup Publication)\}$. Such an entailment should intuitively be pruned in a module that captures the essential meaning of *Car* in the ontology. However, \mathcal{O} also entails that a *Car Owner* owns a *Vehicle*, i.e. $\mathcal{O} \models \{CarOwner \sqsubseteq \exists owns.Vehicle\}$, which is certainly an entailment that should be preserved. Nevertheless, such an entailment is not obtained in Γ_2 .

The example highlights the fact that semantic encapsulation is a necessary but not sufficient condition for capturing the notion of “module”. Intuitively,

a module for an entity should not only encapsulate the entity, but also be coherent and “self-contained” in order to capture adequately its context within the ontology. This idea of self-containment is captured by the notion of *strong encapsulation*

Definition 2 *A component Γ with vocabulary $V_\Gamma = (V_{C_\Gamma}, V_{R_\Gamma}, V_{I_\Gamma})$ is **strongly encapsulating** if it encapsulates all the entities in its vocabulary.*

Strongly encapsulating components represent tightly coupled sub-domains within the ontology they are part of. We will use strong encapsulation to ensure that the relevant components of the ontology can be seen as independent units within the ontology. In the example above, the components Γ_1 , Γ_3 and $\Upsilon = \Gamma_2 \cup \Gamma_3$ are strongly encapsulating.

Note that, in general, there might exist several strongly encapsulating components that encapsulate a given entity in the ontology. In particular, the ontology itself is strongly encapsulating. Intuitively, given an entity we will prefer “smaller” strongly encapsulating components.

Definition 3 *Let Γ, Υ be strongly encapsulating components of \mathcal{O} such that $X \in V_\Gamma$ and $X \in V_\Upsilon$. Then, Γ is **more modular** than Υ with respect to X (denoted by $\Gamma \triangleleft_X \Upsilon$) iff it encapsulates at most as many entities as Υ .*

The definition above suggests that, in order to capture the meaning of a given entity in the ontology, we should find small strongly encapsulating components, since they will preserve the same “essential” information about the entity while ruling out as many undesired entailments as possible.

Definition 4 (minimality)

*A strongly encapsulating component Γ is **minimal** w.r.t. an entity X if $X \in V_\Gamma$ and there is no other component Υ such that $\Upsilon \triangleleft_X \Gamma$.*

Therefore, it seems natural to define a *module* for an entity X as the most modular component w.r.t. X .

Definition 5 (Module) *A module with respect to an entity X is a strongly encapsulating component that is minimal w.r.t. X .*

Therefore, a module for an entity represents a minimal set of axioms that are required to understand, process, evolve and re-use the entity.

In our example Υ is the module for *CarOwner* and *Person*, Γ_3 is the module for *Car* and *Vehicle* and finally Γ_1 is the module for *Publication* and *Book*.

Note that in general there may be several different modules for a given entity within an ontology. For example, suppose that \mathcal{O}_1 and \mathcal{O}_2 are two equivalent ontologies (i.e, they entail each other), and \mathcal{O} is the union of both, then clearly a given entity can have two different modules in \mathcal{O}

The *partitioning* problem is the problem of finding the relevant components in an ontology and can be formulated as follows:

Definition 6 (The Partitioning Problem)

The partitioning problem with input \mathcal{O} is the problem of finding a module associated to each entity in \mathcal{O} .

A module is characterized by two main properties: strong encapsulation and minimality. Any partitioning algorithm should *at least* ensure that the output components are strongly encapsulating, in which case we say that the algorithm is *sound*. If, in addition, the algorithm guarantees minimality we say that it is *optimal*.

In this paper, we make a first step towards the solution of the partitioning problem. The main idea of the method we propose is to transform the original ontology into an \mathcal{E} -Connection with the largest possible number of connected KBs and such that it preserves the semantics of the original ontology in a very specific way. The obtained \mathcal{E} -Connected ontologies are used to generate, for each entity, its \mathcal{E} -module, which is the minimal strongly encapsulating component that can be obtained from the \mathcal{E} -Connection.

The key advantage of relying on \mathcal{E} -Connections is that the *soundness* of the partitioning process can be guaranteed, in the sense that the \mathcal{E} -module for each entity is strongly encapsulating. Moreover, the \mathcal{E} -modules are obtained in a completely automatic way, without relying on heuristics or user intervention of any sort. However, in general, the \mathcal{E} -module for an entity X is not a module for X , in the sense that it is not minimal according to Definition 4. Therefore, we will obtain sub-optimal solutions.

3 \mathcal{E} -Connections of Web Ontologies

In this section, we provide a very brief introduction to \mathcal{E} -Connections. For a detailed discussion we refer the interested reader to [6] [2] and [4].

In a Semantic Web context, an \mathcal{E} -Connection is a set of “ \mathcal{E} -connected” ontologies. From the modeling perspective, each of the ontologies in an \mathcal{E} -Connection is modeling a different application domain, while the \mathcal{E} -Connection itself is modeling the *union* of all these domains. For example, an \mathcal{E} -Connection could be used to model all the relevant information referred to a certain university, and each of its ontologies in the combination could model, respectively, the domain of people involved in the university, the domain of schools and departments, the domain of courses, etc.

Basically, an \mathcal{E} -Connected ontology is an OWL-DL (a.k.a *SHOIN(D)*) ontology extended with the ability to define and use *link properties*. In OWL, object properties (roles, in the DL terminology) are used to relate individuals within a given ontology, while datatype properties relate individuals to data values. In an \mathcal{E} -Connection link properties are used to relate individuals belonging to *different* ontologies in the combination.

As an example, suppose that we want to model the domain of “tourism” as a combination of the following sub-domains: “travel accommodations”, “leisure

activities”, “travel destinations”, and “people”. We want to model each application sub-domain in a different \mathcal{E} -Connected ontology and then use link properties to represent their relationships.

We would like to define classes like *BudgetDestination* (a travel destination which provides a choice of budget accommodations) in the “destinations” ontology, or *CaribbeanHotel* (a hotel accommodation offered at a Caribbean destination) in the “accommodations” ontology.

For such a purpose, we define the link properties *providesAccommodation* and *offersActivity* which relate the domain of “destinations” to the domain of “accommodations” and “activities” respectively, and the link property *hasHobby*, that relates the domain of “people” and the domain of “activities”.

Restrictions on link properties can be used to generate new classes. For example, we can define a “budget destination” in the destinations ontology as a travel destination that offers at least one kind of budget accommodation.

Link properties can be defined as the inverse of another link property, can be tagged as functional or inverse functional and organized in a subsumption hierarchy. However, as opposed to object properties in OWL, a link property cannot be defined as transitive or symmetric.

Formally, we define the language $\mathcal{C}_{\mathcal{IHN}}^{\epsilon}(\mathcal{SHOIN})$ which allows to define combinations of OWL-DL ontologies³ in which inverses, existential, universal and number restrictions as well as hierarchies are allowed on link properties.

Definition 7 (*Syntax of $\mathcal{C}_{\mathcal{IHN}}^{\epsilon}(\mathcal{SHOIN})$*)

Let, for $i, j = 1, \dots, n$ $V_{C_i}, V_{I_i}, \epsilon_{ij}$ be countable and pair-wise disjoint sets of atomic concepts, individuals and property names respectively. The set of ij -properties is $\epsilon_{ij} \cup \{P^- | P \in \epsilon_{ji}\}$. When $i = j$, ij -properties are called “roles”, whereas if $j \neq i$ they are called “links”. The sets of i -concepts are built by simultaneous induction as follows:

$$C := A | \top | \neg D | D \sqcap E | D \sqcup E | \{a\} | \exists P.Z | \forall P.Z | \geq nS | \leq nS$$

Where $A \in V_{C_i}$, D, E are i -concepts, $a \in V_{I_i}$, Z is a j -concept and P, S ij -properties with S simple⁴. An i -axiom \mathcal{A} is an expression of either of the following forms:

$$\mathcal{A} := C \sqsubseteq D | P \sqsubseteq Q | \text{Trans}(R) | C(a) | P(a, b)$$

With P, Q ij -properties, R an ij -property with $i = j$, C, D i -concepts, $a \in V_{I_i}, b \in V_{I_j}$. An \mathcal{E} -Connection \mathcal{K} with vocabulary $V_n = \{\{V_{C_i}\}_{1 \leq i \leq n}, \{V_{I_i}\}_{1 \leq i \leq n}, \{\epsilon_{ij}\}_{1 \leq i, j \leq n}\}$ is a collection $\mathcal{K} = \{\mathcal{K}_1, \dots, \mathcal{K}_n\}$, where \mathcal{K}_i is a finite set of i -axioms.

We don’t provide in this paper a formal discussion of the semantics (see [4],[2] for details), but only the intuition. The main idea is that each ontology in the

³ For brevity, we give here a slightly simplified version of the language that does not capture datatypes. However, all the results presented in this paper also extend for combinations of ontologies with datatypes

⁴ S is *simple* if it is not transitive and none of its sub-properties is transitive

combination is interpreted in a different logical domain, disjoint from the rest. Concepts, roles and individuals are interpreted in their corresponding “local” domain, while link properties act as a bridge by relating elements from different domains. As shown in Definition 7, the \mathcal{E} -Connections formalism imposes certain syntactic restrictions that will be exploited in the partitioning process. For example, a concept in a certain \mathcal{E} -Connected ontology cannot be a subclass of a concept in a different ontology of the combination. These restrictions will guide the partitioning process and will determine to which \mathcal{E} -Connected ontology corresponds each axiom in the original *SHOIN* KB.

4 The Partitioning Algorithm

In this section we show how \mathcal{E} -Connections can be used for solving a simplified version of the partitioning problem. The basic idea is to produce, for each entity in the input ontology, its \mathcal{E} -module. The problem we address can be formulated as follows:

Definition 8 *The \mathcal{E} -Partitioning problem is the problem of finding the \mathcal{E} -module corresponding to each entity in \mathcal{O}*

The generation of the \mathcal{E} -modules from the original ontology can be accomplished by the algorithm presented in Figures 1 and 2. We next explain in detail each step of the algorithm.

1) Safety Check: The safety check is required in order to ensure the correctness of the subsequent steps. Intuitively, 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 9 (safety)

Let: $g : C \in \mathcal{O} \rightarrow \{T, F\}$ be a function mapping every 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 $g(D) = T$;*

A detailed discussion of the notion of safety can be found in [3]

2) Generation of the \mathcal{E} -Connection: In case of a positive result in the safety check, the algorithm generates in this step the \mathcal{E} -Connection $\mathcal{K} = \{\mathcal{K}_1, \dots, \mathcal{K}_n\}$ using the algorithm in Figure 1. The algorithm performs a succession of n *partitioning steps*. Each step involves a *pair* of KBs: the KB, \mathcal{K}_0 , that is initially set to the original ontology \mathcal{O} and from which entities and axioms are removed, and a *target* KB, \mathcal{K}_i , generated from scratch in the i th step, to which these are added. In the process some roles in \mathcal{K}_0 will eventually become link properties.

In a given partitioning step, each concept, individual and link property (generated in a previous step) can be in one of two possible “states”: either as entities in \mathcal{K}_0 , or in \mathcal{K}_i . A role, however, can be in one of four possible states: as a role in \mathcal{K}_0 (State 1), as a link property from \mathcal{K}_0 to \mathcal{K}_i (State 2), as a link property from \mathcal{K}_i to \mathcal{K}_0 (State 3), and finally as an object property in \mathcal{K}_i (State 4). Only the transitions $1 \rightarrow 2$, $1 \rightarrow 3$, $1 \rightarrow 4$, $2 \rightarrow 4$ and $3 \rightarrow 4$ are allowed.

The algorithm starts a partitioning step by creating a new component \mathcal{K}_i and by forcing an initial state transition on an arbitrary entity in \mathcal{K}_0 . The initial transition will trigger new ones, due to the syntactic constraints imposed by \mathcal{E} -Connections. For example, if $(C \sqsubseteq D) \in \mathcal{K}_0$, and we move C , then D *must* be exported as well to \mathcal{K}_i , since an axiom cannot relate complex classes in different components in an \mathcal{E} -Connection. However, there is still a choice to make in certain situations that involve roles. For example, if $\exists R.C \in \mathcal{K}_0$ and we move C , two possible actions would be allowed: first, to make R a Link Property from \mathcal{K}_0 to \mathcal{K}_i ; second, to make R a role in \mathcal{K}_i . Analogously, suppose that $P(a, b) \in \mathcal{K}_0$, and we move a ; again, two choices would be admissible: either we make P a Link Property from \mathcal{K}_i to \mathcal{O} , or we make it a role in \mathcal{K}_i . In both examples, each choice would result in a syntactically valid \mathcal{E} -Connection. In order to obtain a maximal partitioning of \mathcal{O} we will transform roles into link properties *whenever possible*.

The set $\text{bounded}(P)$ represents the set of entities that are “forced” to end up in the same \mathcal{E} -Connected ontology due to the semantics of the link property P . For example, imagine we have 3 \mathcal{E} -Connected KBs: $\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_2$. Suppose we first create \mathcal{K}_1 and then \mathcal{K}_2 . Suppose that there is a Link Property P , generated in the first step, from \mathcal{K}_1 to \mathcal{K}_0 and $\exists P.C, \exists P.B \in \mathcal{K}_0$. Then, assume that C is moved from \mathcal{K}_0 to \mathcal{K}_2 in the second step. Obviously, P should now link \mathcal{K}_1 and \mathcal{K}_2 , instead of \mathcal{K}_1 and \mathcal{K}_0 , and hence B must be moved as well, since it is bounded to C by P .

Once all the state transitions in a partitioning step have been completed, the relevant axioms are moved. Each axiom in the input ontology is moved *only once*, i.e. whenever it is exported to a newly created component, it will never be put back, nor moved to a different component. As an example, let us consider the following KB:

$$\mathcal{O} = \{(C \sqsubseteq D \sqcup E); (C \sqsubseteq \neg D); (B \sqsubseteq \exists P.A); (A \sqsubseteq F)\}$$

The KB is clearly safe. The reader can verify using Figure 1 that we finally obtain the \mathcal{E} -Connection $\mathcal{K} = \{\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3\}$, with:

$$\mathcal{K}_1 = \{(C \sqsubseteq D \sqcup E); (C \sqsubseteq \neg D)\}; \mathcal{K}_2 = \{(A \sqsubseteq F)\}; \mathcal{K}_3 = \{(B \sqsubseteq \exists P.A)\}$$

```

-Algorithm GenerateE-Connection( $\mathcal{O}$ )
-Input: A SHOIN ontology  $\mathcal{O}$ 
-Output: A maximal  $\mathcal{E}$ -Connection  $\mathcal{K} = \{\mathcal{K}_1, \dots, \mathcal{K}_n\}$ 

 $\mathcal{K} \leftarrow \{\mathcal{K}_0\}$  with  $\mathcal{K}_0 = \mathcal{O}$ 
 $S \leftarrow$  Set of all concepts, roles and individuals in  $\mathcal{O}$ 
if  $\mathcal{O}$  not  $\mathcal{E}$ -safe, return  $\mathcal{K} = \{\mathcal{K}_0\}$ 
for each  $X \in S$ ,  $State(X) \leftarrow 1$ 
Repeat   Select  $X \in S$  with  $State(X) = 1$ 
  if  $X$  a concept, individual or link property, then  $State(X) \leftarrow 2$ 
  if  $X$  a role then  $State(X) \leftarrow 3$ 
  Create new KB  $\mathcal{K}_i$ .  $\mathcal{K} \leftarrow \mathcal{K} \cup \{\mathcal{K}_i\}$ 
  Repeat
    for all concept  $C \neq \top$  with  $State(C) = 1$ 
      if any of the following conditions holds:
        1)  $(C \sqsubseteq D)$  or  $(D \sqsubseteq C) \in \mathcal{O}$ , and  $State(D) = 2$ 
        4)  $C(a) \in \mathcal{O}$  and  $State(a) = 2$ 
        5)  $\exists P.C$  or  $\forall P.C \in \mathcal{O}$ , for  $P$  a role, and  $State(P) \in \{2, 4\}$ 
        6)  $C \sqcap D$  ( $C \sqcup D$ )  $\in \mathcal{O}$  and  $State(D) = 2$ 
        7)  $C = D \sqcap E$ , or  $C = D \sqcup E$  and  $State(E) = 2$  or  $State(D) = 2$ 
        8)  $C$  a restriction  $\exists P.D, \forall P.D, \geq nP$  or  $\leq nP$ ,
           with  $P$  a role and  $State(P) \in \{3, 4\}$ , or a link with  $State(P) = 2$ 
        9)  $C = \{a\}$  and  $State(a) = 2$ 
        10)  $(\neg C) \in \mathcal{O}$  and  $State(\neg C) = 2$ 
        10)  $C, E \in Bound(P)$  and  $State(E) = 2$ 
      then  $State(C) \leftarrow 2$ 
    for all individual  $a$  with  $State(a) = 1$ 
      if any of the following conditions holds:
        1)  $C(a) \in \mathcal{O}$  and  $State(C) = 2$ 
        2)  $\{a\} \in \mathcal{O}$  and  $State(\{a\}) = 2$ 
        3)  $P(a, b) \in \mathcal{O}$  and either of the following holds:
          3.1)  $P$  a role with  $State(P) = 3$  or  $State(P) = 4$ 
          3.2)  $P$  a Link Prop. with  $State(P) = 2$ 
          4)  $P(b, a) \in \mathcal{O}$ ,  $P$  a role, and  $State(P) = 2$  or  $State(P) = 4$ 
        then  $State(a) \leftarrow 2$ 
    for all Link Prop.  $P$  with  $State(P) = 1$ 
      if any of the following conditions holds:
        1)  $(P \sqsubseteq Q)$  or  $(Q \sqsubseteq P) \in \mathcal{O}$ , and  $State(Q) = 2$ 
        3) There is a restriction  $D$  with property  $P$  and  $State(D) = 2$ 
        4)  $P(a, d) \in \mathcal{O}$  and  $State(a) = 2$ 
      then  $State(P) \leftarrow 2$ 
    for all role  $P$  with  $State(P) \in \{1, 2, 3\}$ 
      if  $(P \sqsubseteq Q)$  or  $(Q \sqsubseteq P) \in \mathcal{O}$ , and  $State(Q) \in \{2, 3, 4\}$  then  $State(P) \leftarrow State(Q)$ 
      if  $State(P) \in \{2, 3\}$  and  $P$  transitive, then  $State(P) \leftarrow 4$ 
      if  $State(P) = 1$ ,  $P(a, b) \in \mathcal{O}$  and  $State(a) = 2$ , then  $State(P) \leftarrow 3$ 
      if  $State(P) = 2$ ,  $P(a, b) \in \mathcal{O}$  and  $State(a) = 2$ , then  $State(P) \leftarrow 4$ 
      if  $State(P) = 1$ ,  $P(a, b) \in \mathcal{O}$  and  $State(b) = 2$ , then  $State(P) \leftarrow 2$ 
      if  $State(P) = 3$ ,  $P(a, b) \in \mathcal{O}$  and  $State(b) = 2$ , then  $State(P) \leftarrow 4$ 
      if  $State(P) = 1$ ,  $D$  a restr. on  $P$ ,  $State(D) = 2$ , then  $State(P) \leftarrow 3$ 
      if  $State(P) = 2$ ,  $D$  a restr. on  $P$ ,  $St(D) = 2$ , then  $St(P) \leftarrow 4$ 
      if  $State(P) = 1$ ,  $\exists P.C$  or  $\forall P.C \in \mathcal{O}$ ,  $State(C) = 2$ , then  $State(P) \leftarrow 2$ 
      if  $State(P) = 3$  and  $\exists P.C$  or  $\forall P.C \in \mathcal{O}$  with  $State(C) = 2$ , then  $State(P) \leftarrow 4$ 
      if  $State(P) = 1$  and  $State(Inv(P)) = 2$ , then  $State(P) \leftarrow 3$ 
      if  $State(P) = 1$  and  $State(Inv(P)) = 3$ , then  $State(P) \leftarrow 2$ 
      if  $State(P) = 1, 2$ , or  $3$  and  $State(Inv(P)) = 4$ , then  $State(P) \leftarrow 4$ 
    until no state transition has occurred
  for each Axiom  $\mathcal{A} \in \mathcal{O}$ 
    if any of the following conditions holds
      1)  $\mathcal{A}$  of the form  $C \sqsubseteq D$ ,  $State(C) = State(D) = 2$ 
      2)  $\mathcal{A}$  of the form  $C(a)$ ,  $State(a) = State(C) = 2$ 
      3)  $\mathcal{A}$  of the form  $R_1 \sqsubseteq R_2$ , and either  $State(R_1) = 3$ ,
          $State(R_2) = 3$ , or  $State(R_1) = 4$ ,  $State(R_2) = 4$ 
      4)  $\mathcal{A}$  of the form  $Trans(R)$ ,  $State(R) \in \{3, 4\}$ 
      5)  $\mathcal{A}$  of the form  $R(a, b)$ ,  $State(R) \in \{3, 4\}$ 
      6)  $\mathcal{A}$  of the form  $G_1 \sqsubseteq G_2$ ,  $State(G_1) = State(G_2) = 2$ 
      7)  $\mathcal{A}$  of the form  $G(a, b)$ ,  $State(G) = 2$ 
    then  $\mathcal{O} \leftarrow \mathcal{O} - \{\mathcal{A}\}$  and  $\mathcal{K}_i \leftarrow \mathcal{K}_i \cup \{\mathcal{A}\}$ 

  for each  $P$ , s.t.  $State(P) = 2$ , make  $P$  a link prop. with  $State(P) \leftarrow 1$ 
  for each link prop.  $P \in \mathcal{K}_j$ ,  $\mathcal{K}_j \in \mathcal{K}$ , and  $\mathcal{K}_j \neq \mathcal{K}_i$ 
    if  $P$  points to  $\mathcal{K}_0$  and  $\forall X \in bounded(P)$ ,  $State(X) = 2$ , make  $P$  point to  $\mathcal{K}_i$ 
  until  $\mathcal{K}_0 = \emptyset$ 
 $\mathcal{K} \leftarrow \mathcal{K} - \mathcal{K}_0$ 
return  $\mathcal{K}$ 

```

Fig. 1. Steps 1) and 2) of the Partitioning Algorithm

Where P is a link property from \mathcal{K}_3 to \mathcal{K}_2 .

3) Generation of the \mathcal{E} -modules: The input to this step is the \mathcal{E} -Connection expressed in the logic $\mathcal{C}_{\mathcal{IHLN}}^\epsilon(\mathcal{SHOIN})$. The output is the \mathcal{E} -module corresponding to each entity in \mathcal{O} . Please, note that an \mathcal{E} -module is just a component of \mathcal{O} and hence it is expressed as an ordinary \mathcal{SHOIN} KB and *not* as \mathcal{E} -Connection. \mathcal{E} -Connections are *transparent* to the user, since both the input and the output of the problem are represented in plain OWL.

From the \mathcal{E} -Connection it is possible to define the *partitioning graph* of the ontology.

Definition 10 (*Partitioning Graph for \mathcal{O}*)

Let $\mathcal{K} = \{\mathcal{K}_1, \dots, \mathcal{K}_n\}$ be the \mathcal{E} -Connection obtained from \mathcal{O} . The partitioning graph $G(\mathcal{O}) = (V, E)$ is a Simple Directed Graph defined as follows:

- The node $v_i \in V$ is the set of axioms contained in \mathcal{K}_i where the link properties in \mathcal{K}_i have been transformed into ordinary roles in v_i .
- The pair (v_i, v_j) is an edge in E if there is an ij -property in \mathcal{K} with $i \neq j$.
- Each edge (v_i, v_j) is labeled with the set of ij -properties in \mathcal{K}

The nodes of the graph are subsets of the axioms in the original ontology. The information about the link properties in the \mathcal{E} -Connection is stored in the edges of the graph. The nodes of the graph do not share axioms, but may share entities in their respective vocabularies. If an entity X belongs to the vocabulary of a single node v_k , we say that X corresponds to v_k . If the entity is used in a set v_1, \dots, v_j of nodes, then the syntax of \mathcal{E} -Connections ensures that there will be only one node $v_i, 1 \leq i \leq j$ in which the entity will not appear within the scope of a quantifier. We then say that X corresponds to v_j .

From the partitioning graph the \mathcal{E} -modules for each entity are obtained (Figure 2).

Definition 11 (*\mathcal{E} -module*)

Let $G(\mathcal{O})$ be the partitioning graph for \mathcal{O} and let $X \in \mathcal{O}$ be an entity corresponding to the node $v_i \in G(\mathcal{O})$. The \mathcal{E} -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 $G(\mathcal{O})$.

By construction, given two entities, their respective \mathcal{E} -modules coincide iff they correspond to the same node in $G(\mathcal{O})$.

The fundamental advantage for using \mathcal{E} -Connections as a partitioning tool is characterized by the following result:

Theorem 1 (*Correctness*) *An \mathcal{E} -module is strongly encapsulating*

Proof (Sketch): Let X be an arbitrary entity encapsulated by the \mathcal{E} -module Γ . The fact that all the atomic entailments involving X that hold in Γ will also hold in \mathcal{O} follows directly from the monotonicity of \mathcal{SHOIN} , since $\Gamma \subseteq \mathcal{O}$. The proof in the other direction is rather involved and requires a thorough analysis of

```

-Algorithm GenerateModules( $G(\mathcal{O})$ )
-Input: The partition graph  $G(\mathcal{O})$ 
-Output: A function  $f$  mapping each entity in  $\mathcal{O}$  to its  $\mathcal{E}$ -module

-for each node  $v_i$  in  $G(\mathcal{O})$ 
   $\mathcal{Y} \leftarrow v_i$ 
  Do a BFS search from  $v_i$  and add to  $\mathcal{Y}$  all the axioms in the visited nodes
  for each entity  $X$  corresponding to  $v_i$ , do  $f(X) = \mathcal{Y}$ 

```

Fig. 2. Step 3) of the Partitioning Algorithm

the relation between the semantics of the original ontology and the \mathcal{E} -Connection \mathcal{K} obtained in Step 2). We give here the main idea and refer to the interested reader to [3] for more details. The \mathcal{E} -Connection \mathcal{K} contains, by construction, exactly the same axioms as \mathcal{O} . However, the semantics of \mathcal{E} -Connections differs from the semantics of ordinary DLs. The safety check in Step 1) and the way \mathcal{K} has been built in Step 2) ensure that all the atomic entailments involving X in \mathcal{O} will hold in \mathcal{K} and also in any \mathcal{E} -Connection \mathcal{K}^* obtained by collapsing several \mathcal{E} -Connected ontologies in \mathcal{K} into a single one (this operation is called a merge [3]). Given X , we can always find a merge s.t., for $\mathcal{K}^* = (\mathcal{K}_1^*, \dots, \mathcal{K}_m^*)$, the \mathcal{E} -module of X coincides with some K_i^* , $1 \leq i \leq m$ and such that K_i^* contains no link properties. Therefore, since all the atomic entailments for X hold in \mathcal{K}^* , they will also hold in $\Gamma = \mathcal{K}_i^*$. **Q.E.D**

The following lemma shows that the \mathcal{E} -module of a given entity is the minimal self-contained component we can obtain using \mathcal{E} -Connections.

Lemma 1 *Given the entity X , its \mathcal{E} -module is the minimal strongly encapsulating component that can be obtained from the partitioning graph s.t. it encapsulates X .*

The proof is a rather straightforward consequence of the fact that \mathcal{K} is the \mathcal{E} -Connection with the largest number of components we can obtain (See [3]). In general, if Γ is the \mathcal{E} -module for X , and Σ its module, the following will hold:

$$\Sigma \triangleleft_X \Gamma \triangleleft_X \mathcal{O}$$

In other words, the \mathcal{E} -module for X is less modular than the actual module for X but more modular than the ontology as a whole.

The following lemma shows that the \mathcal{E} -partitioning problem can be solved efficiently:

Lemma 2 *The algorithms described in Figure 1 and 2 are worst-case quadratic in the size of the input ontology*

Steps 1) and 2) are proved to be quadratic in [3]. The complexity of Step 3) depends on the complexity of Breadth-First Search (BFS) in simple directed graphs, which is linear in the size of $G(\mathcal{O})$ (number of nodes plus number of

edges). Since we perform as many BFSs as nodes in $G(\mathcal{O})$, Step 3) is quadratic in the size of the graph. Note that the number of nodes and edges in $G(\mathcal{O})$ is bounded by the number of axioms and the number of link properties in \mathcal{O} respectively and hence the size of $G(\mathcal{O})$ is bounded by the size of \mathcal{O} .

Finally, note that the partitioning algorithm presented in this section is *completely automatic*. No user intervention, nor any heuristic, is required at any stage of the process.

4.1 How to use the modules

The simplest way to publish a modular ontology is to publish all the modules as distinct OWL documents, with their own, coined, URI. Each module could then be imported in isolated by users⁵. Given the efficiency of the partitioning algorithm and the slow rate of change of most ontologies, regenerating the partitions would not be burdensome. However, the URIs should be carefully designed to indicate the connection with the overall ontology as well as the entities encapsulated by the module.

Alternatively, one could design a suitable XPointer scheme⁶ to represent various modules of an entity in an ontology. For example, an XPointer for \mathcal{E} -modules of a class might look as follows:

`http://example.org/aLargeOntology#emoduleFor(aClass)`⁷

The fragment identifier `emoduleFor(aClass)` is defined to identify the \mathcal{E} -module for `aClass`, and, if the object of an *owl:imports* statement, would require the client to partition the retrieved OWL document and only import the specified part of the ontology. With server-side XPointer[1], partitioning can be left entirely to the server which can choose to preprocess the ontology into a number of documents, or generate the module on the fly. In either case, the XPointer scheme provides a clear, extensible, standard way for referring to principled subparts of OWL ontologies.

5 Implementation and Discussion

We have implemented the partitioning algorithm on top of Manchester's OWL-API, which we have extended to provide support for \mathcal{E} -Connections. The UI in SWOOP⁸ for browsing \mathcal{E} -Connections has been extended to support automated partitioning. We have applied our algorithm to a set of OWL ontologies available on the Web. Table 2 summarizes the results obtained for some relevant cases.

⁵ The use of the *owl:imports* construct requires the imported ontology to be designated using a URI

⁶ <http://www.w3.org/TR/WD-xptr>

⁷ For brevity, we presume a W3C developed XPointer scheme, thus obviating the need for an embedded namespace URI.

⁸ <http://www.mindswap.org/2004/SWOOP>

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

The OWL-S ontologies describe Web services. Each ontology in the set models a well-defined sub-domain, while the Web Services domain as a whole is modeled using *owl:imports* profusely. We have collapsed the set of ontologies respecting the semantics of *owl:imports* and applied the partitioning algorithm to the result. The partitioning graph is shown in Figure 3. The green nodes in the graph represent completely independent subsets of the original ontology; the blue ones (*leaf* nodes) represent components that are “used” by other components but which do not rely on any other component in the graph. The \mathcal{E} -module for an entity corresponding to a green or a blue node will contain *only* the axioms corresponding to the node, and hence we can say that green and blue nodes are “free-standing”. Finally, in red, we have identified the nodes with outgoing edges. Consider the entity X corresponding to the red node in the sub-graph highlighted in Figure 1. The \mathcal{E} -module for X is obtained as the union of the axioms corresponding to the 3 nodes highlighted in the Figure.

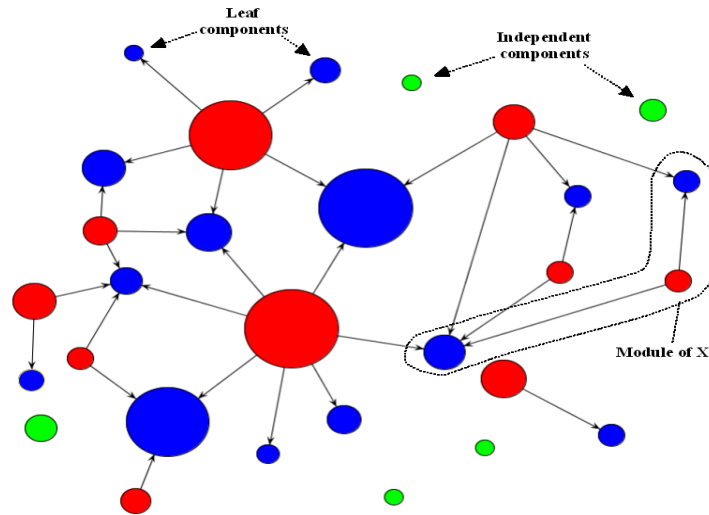


Fig. 3. Partitioning Graph for OWL-S

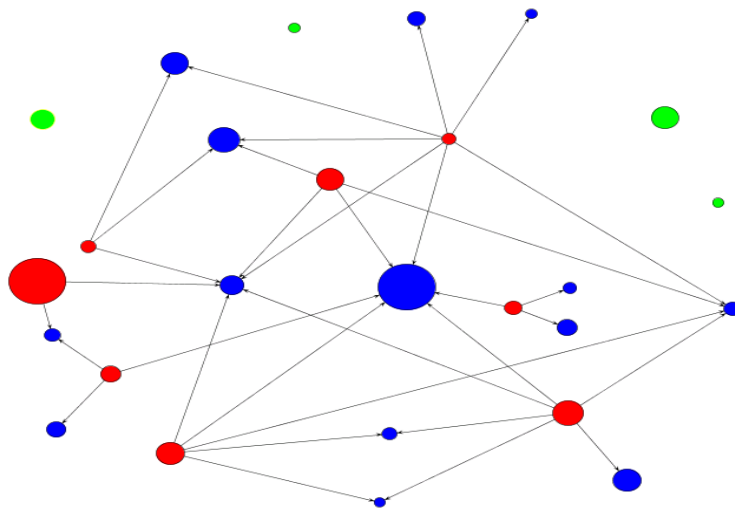


Fig. 4. Partitioning Graph for NCI

NCI is a huge, carefully designed, ontology dealing with the biomedical domain. The partitioning graph for NCI is shown in Figure 4. The graph also exhibits a nice decomposition. An important proportion of nodes correspond to independent or leaf components. Interestingly, for both OWL-S and NCI, there is an improvement in modularity for every entity, in the sense that every \mathcal{E} -module is *strictly smaller* than the ontology as a whole.

However, there are some ontologies that cannot be modularized. The medical ontology GALEN is a prominent example. In GALEN, every entity (except for one class) is ultimately depending on a common *top* class and *top* role. Hence, although it is possible to identify intuitively several disjoint sub-domains, the ontology follows a very “monolithic” design pattern, which prevents a good partitioning.

Finally, partitioning can reveal modeling flaws in the input ontology. This is the case, for example of NASA’s SWEET-JPL ontologies. In SWEET-JPL, the domain is modeled using several ontologies interrelated using *owl:imports*. The *owl:imports* construct has been used to “modularize” the domain. However, such an *ad-hoc* modularization turns out to be wrong. Our partitioning shows that the physically distinct ontologies do not correspond at all to semantically encapsulating components.

6 Related Work

Partitioning OWL ontologies for modeling purposes has been recently discussed in [9] and [8]. We consider that the crucial limitation of both works is that

they do not provide a definition of the problem to be solved. In [9], the output is just a graph visualization of the different kinds of information contained in the input ontology. There is no correspondence between the nodes of the graph and a set of axioms. Moreover, the heuristics considered in [9] to generate the visualization only consider a tiny subset of OWL, which basically corresponds to RDFS. The PromptFactor tool which uses structural tracing for extracting relevant sub-parts of ontologies is described in [8]. Although the output in this case is a set of axioms, there is no formal characterization of the relationship between the semantics of a given term in the input ontology and in the output fragment and hence no notion of correctness of the process is established.

Finally, [7] 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.

7 Conclusion and Future Work

In this paper, we have addressed the problem of determining the subset of an ontology that captures the essential meaning of a given entity. We have provided a formal framework for understanding the notion of modularity in a Semantic Web context and provided an algorithm, based on \mathcal{E} -Connections, for identifying modules within an ontology, which are expressed in plain OWL. Our experimental results show that for a significant number of entities the obtained modules can be notably smaller than the original ontology, which facilitates re-use, processability, understandability and maintenance.

References

1. Kendall Clark, Bijan Parsia, Bryan Thompson, and Brad Bebee. A semantic web resource protocol: Xpointer and http. In *Proceedings of ISCW 2004*, 2004.
2. B. Cuenca-Grau, B. Parsia, and E.Sirin. Combining owl ontologies using e-connections. Technical report, UMIACS, 2004. To Appear in Journal of Web Semantics.
3. B. Cuenca-Grau, B. Parsia, E.Sirin, and A.Kalyanpur. Automatic partitioning of owl ontologies using e-connections. Technical report, UMIACS, 2005. Available at <http://www.mindswap.org/2004/multipleOnt/papers/Partition.pdf>.
4. B. Cuenca-Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *Proc. of ISWC 2004*, 2004.
5. A. Kalyanpur, B. Parsia, and J. Hendler. A tool for working with web ontologies. *Int. J. on Semantic Web and Info. Syst.*, 1(1), 2004.
6. O. Kutz, C. Lutz, F. Wolter, and M. Zakharyashev. E-connections of abstract description systems. *Artificial Intelligence* 156(1):1-73, 2004.
7. 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.
8. N.F. Noy and M.A. Musen. The prompt suite: Interactive tools for ontology mapping and merging. *Int.J. of Human-Computer Studies*, 6(59), 2003.
9. H. Stuckenschmidt and M. Klein. Structure-based partitioning of large class hierarchies. In *Proc. of ISWC 2004*, 2004.