Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective

Paul Doran
Dept. of Computer Science,
University of Liverpool, UK
pdoran@csc.liv.ac.uk

Valentina Tamma
Dept. of Computer Science,
University of Liverpool, UK
valli@csc.liv.ac.uk

Luigi Iannone
Dept. of Computer Science,
University of Liverpool, UK
luigi@csc.liv.ac.uk

ABSTRACT

Problems resulting from the management of shared, distributed knowledge has led to ontologies being employed as a solution, in order to effectively integrate information across applications. This is dependent on having ways to share and reuse existing ontologies; with the increased availability of ontologies on the web, some of which include thousands of concepts, novel and more efficient methods for reuse are being devised. One possible way to achieve efficient ontology reuse is through the process of *ontology module extraction.* A novel approach to ontology module extraction is presented that aims to achieve more efficient reuse of very large ontologies; the motivation is drawn from an Ontology Engineering perspective. This paper provides a definition of ontology modules from the reuse perspective and an approach to module extraction based on such a definition. An abstract graph model for module extraction has been defined, along with a module extraction algorithm. The novel contribution of this paper is a module extraction algorithm that is independent of the language in which the ontology is expressed. This has been implemented in ModTool; a tool that produces ontology modules via extraction. Experiments were conducted to compare ModTool to other modularisation methods.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation languages;
I.2.6 [**Learning**]: Knowledge acquisition

General Terms

Algorithms

Keywords

ontology module extraction, ontology engineering

1. INTRODUCTION

Ontologies have been successfully employed in order to solve problems deriving from the management of shared, distributed knowledge, and the efficient integration of information across applications [7]. Much of this success depends on the ability to share and reuse existing ontologies [10]. Ontology construction is deemed to be a time consuming and labour intensive task, but is mediated by the possibility of reusing existing ontologies. This is greatly facilitated by the existance of Ontology Libraries (for example, the DAML Ontology Library[1]), and the emergence of search engines such as Swoogle[2] and Ontosearch[3] which support the retrieval of web ontologies.

Many ontology development methods and methodologies, such as the Ontology 101 method [15] and Methontology [11], include a reuse step in the ontology development lifecycle that allow Ontology Engineers to integrate into the ontology they are currently designing and implementing an ontology that has already been developed[4]. The reuse of existing ontologies can occur at the design stage and at the implementation stage. Some ontology editors, for example Protégé [8], allow the reuse of another ontology by including it in the model that is being designed. In Protégé this happens through the inclusion of other projects, which operates at the knowledge level. The web ontology language OWL[5] offers the possibility of importing an OWL ontology by means of the `<owl:imports>` statement, and many ontology development tools allow the import of OWL files. In both cases, the whole ontology is included, which can create a huge overheard. However, ontology developers might only be interested in a portion of the original ontology, especially when the ontology being reused is very large. For example, a developer might only be interested in concepts about and relating to diabetes; but they have to import a whole medical ontology, such as UMLS [6], in order to get what was required. Currently there are no methods that allow only part of an ontology to be specified and reused.

Ontology modularisation [17] could help to overcome the problem of identifying a fragment of an existing ontology to be reused, in order to enable ontology developers to include

[1]http://www.daml.org/ontologies/
[2]http://swoogle.umbc.edu/
[3]http://www.ontosearch.org/
[4]For an extensive review of the state of the art on Ontology Engineering see [10]
[5]http://www.w3.org/2004/OWL/#specs
[6]An OWL translation is available from:
http://swpatho.ag-nbi.de/owldata/ swpatho1/umlssn.owl

only those concepts and relations that are relevant for the application they are modelling an ontology for. Ontology Engineering good practices list modularisation as one of the principles for building good ontologies [1]. Whilst it is possible to build modular ontologies in OWL or RDF there are no explicit constructs for defining module boundaries. Indeed such constructs may not be that useful because the module boundaries they define may be inappropriate for another ontologies application.

In addition to reuse, ontology modularisation has other applications. These include query answering, distributed reasoning; and scalable evolution and maintenance. Each application places different requirements upon the ontology modularisation process. This paper's focus is on ontology module extraction for reuse by providing two main contributions: we investigate and demonstrate a method for reusing ontologies that depends only on the ontological model, and is independent of the specific formalism used to represent the ontology by producing highly reusable entities in the form of ontology modules. In addition, we discuss the phases that underlie the successful identification, and reuse of an ontology module.

The paper is organised as follows. Section 2 reviews the different approaches to ontology modularisation. Section 3 lays out a framework for ontology modularisation. A definition of an ontology module is provided. Section 4 details a proposed methodology for reusing ontology modules. In Section 5 an abstract graph model for module extraction is presented. This model can be applied to RDFS and OWL. Particular attention is paid to the rules used to generate the ontology module, as these guarantee transitive closure. Section 6 details the experiments conducted up to this point. A discussion of the results is provided and these show that ModTool compares favourably with current approaches. Finally, the conclusions are presented in Section 7; along with possible directions for future work.

2. RELATED WORK

In this section a review and critical discussion of the current approaches to ontology modularisation is carried out. The problem of dividing an ontology into a number of modules has received much attention in recent years. A number of approaches that aim at solving this problem have been proposed, such as the approach by Noy and colleagues [16] to determine ontology views, or the partitioning algorithm by Cuenca Grau and colleagues [4]. Ontology modularisation is also the focus of collaborative research in work package 2.1 of the EU funded network of excellence Knowledge Web[7]. This has already produced an extensive overview on ontology modularisation in the deliverable D2.1.3.1 [17]; this is followed in order to define ontology modularisation and its possible uses.

The method by Stuckenschmidt & Klein[22] consists of automatically partitioning large ontologies into smaller partitions based on the structure of the class hierarchy; taking into account the internal coherence of the concepts. The method can be broken down into two tasks; the creation of a weighted graph and the identification of partitions from the dependency graph. The weights defined in the first step determine the results of the second step.

Noy & Musen[16] create a 'view' on the ontology, analogous to views in databases. The method is integrated into Protégé. The view is produced by traversal of relations that are specified by the user. The user selects the starting point for the traversal and sepcifies which relations are traversed and for how far. The result of this traversal is a view on the ontology; the view retains all its relations to the source ontolgoy via the 'view boundary'. The view is not extracted nor is it treated as a separate entity. This allows the user to shrink or expand the view as required. The focus of this method is on query answering rather than reuse.

Cuenca Grau *et al.*s [4, 12] cast the problem of modularisation as a problem of partitioning an ontology *safely.* Cuenca Grau *et al.*provide an operational definition for characterising 'safely partitionable ontologies', though their work fails to provide a definition of what a module is. An intuitive notion of module is provided due to the fuzzy nature of the modularisation problem. They propose then an algorithm for obtaining such partitions whose elements are disjunct w.r.t. the axioms contained. This approach is theoretically rigourous, but this rigour reduces the possibility for reuse because the method cannot be applied to any ontology. The ontology must pass a safety check before the method can be applied. This is not ideal as the number of possible ontologies available for reuse is reduced.

Cuenca Grau *et al.*work is based on the notion of conservative extensions [9]. This means that essential inferences about the entities contained within an element of such a partition should be preserved. Whilst conservative extensions can theoretically be used to define an ontology module, they cannot currently be used in practice as deciding if an OWL-DL module is a conservative extension is undecidable [14]. However, conservative extensions can be used with less expressive languages to help in the construction of a modular ontology.

Seidenberg & Rector[20] produce ontology segments about one or more classes of the userÕs choice; the algorithm bases the extraction on these and related concepts. Whilst the algorithm produces desirable results on Galen it is difficult to see how the algorithm can be made general purpose due to the number of parameters that have to be set.

D'Aquin *et al.*[5] define a sub-vocabulary of the ontology that the ontology module should describe; for an element to be included in the ontology module certain criteria have to be met. The main criticism of this approach is that it is tightly focused on knowledge selection. However, it would seem that the algorithm could be deployed on different modularisation applications.

There is also the question of user involvement. Depending on the context of the reuse, user involvement may be beneficial. When reuse is required online user involvement should be kept to a minimum, but if reuse is being carried out offline then the benefits of user involvement could be capitalised upon. Both Stuckenschmidt and Klein, and Cuenca Grau *et al* methods are automatic and require minimal user involvement. This makes them fairly simple to use, but there is little control over the results. Noy and Musen's method requires user involvement and the results of the method can therefore be tailored by the user to their exact requirements.

Stuckenschmidt and Klein do not consider the semantics of the relations in their method and instead offer a "simple and scalable method based on the structure of the ontology"[22]. This is a drawback to their approach because con-

[7]http://knowledgeweb.semanticweb.org/

sidering what the relations 'mean' is important in determining modules. This is why the rules applied by the approach in this paper independently consider the different types of relations. Noy and Musen allow the user to state what relations are to be considered, however their method is based on RDFS; and its extension to OWL is not described, but the authors say it can be extended. Whilst Cuenca Grau *et al* method applies to OWL and also considers, at some level, the semantics. However ε-connections have limited expressiveness of the connections made between partitions because they cannot express subclass or sub-properties relations.

Lastly, Stuckenschmidt and Klein and Cuenca Grau *et al* are 'one shot' approaches. The process is run on the ontology and the results are obtained. In certain circumstances this may be sufficient, but it is not hard to imagine circumstances where the results obtained are not optimal. For example, consider the case of a large ontology which would most likely produce large partitions. Whereas, Noy and Musen allow the user to run the method with different configurations to refine the results to their requirements.

In this paper we present a novel approach to ontology module extraction that aims to produce highly reusable entities in the form of ontology modules. Rector [18] state that it is necessary that the modules to be reused can be identified and separated from the whole. This led us to pursue module extraction from the pragmatic stand point of an Ontology Engineer. Allowing the Ontology Engineer to have some control over the process allows them to identify which part of the ontology they wish to separate from the whole. Our approach exploits an abstract graph model of the ontology rather than its representation in a logical formalism like DL that underlies OWL, thus making it more general and applicable to ontologies written in various formalisms, from RDF to OWL-DL. The implementation of ModTool was based on the abstract graph model given in Section 5.

Additionally, the approach presented is based on the notion of the modularisation process being user-led. This is because the Ontology Engineer needs to select which concept they want the ontology module to be about. In addition, there was a need for the ontology module produced to be user customisable. This makes the ontology modules highly reusable because the Engineer can modify the module to conform to their conceptualisation of the domain. For example adding equivalent classes to conform to the natural language commitments of the developers application. ModTool emphasises that the output produced has the least constraints imposed upon it as possible. This allows the user to tailor it to their specific individual requirements.

The ModTool approach makes several contributions, the first being that it produces highly reusable entities in the form of ontology modules. The pragmatic approach taken from the view of an Ontology Engineer introduces a different perspective of ontology modularisation that is introduced in Section 3.

3. FRAMEWORK FOR ONTOLOGY MODULE EXTRACTION

Before proceeding to describe the approach to ontology module extraction currently being investigated, we clarify what an ontology module is in the context of this approach. Several definitions exist within the literature. By formulating a possible definition of an ontology module it is necessary to reconcile it with the previous definitions. It is hoped that this will speed up the process of building toward a consensus.

While the notion of module is quite well understood and accepted in the area of Software Engineering, it is not clear at all what the characteristics of an ontology module are [22]. For the purpose of this paper an ontology module is the following:

> *An ontology module is a reusable component of a larger or more complex ontology, which is self-contained but bears a definite association to other ontology modules, including the original ontology.*

This definition implies that ontology modules can be reused by developers either as they are, or by extending them with new concepts and relationships. The process of module extraction makes this possible because it takes existing ontologies and produces ontology modules. If ontology modules can be extended with new concepts and relationships then each ontology module should be viewed as an ontology itself. This notion seems intuitive.

The validity of this definition needs to be assessed in relation to the literature. Cuenca Grau *et al* [3] state "a module should contain information about a self-contained subtopic." This concurs with Stuckenschmidt and Klein [21] who state "in order to facilitate the reuse of individual modules we have to make sure that modules are self-contained." Thus, an ontology module is an ontology that is a self-contained subset of a parent ontology. An ontology module is self-contained if all the concepts defined in the module are defined in terms of other concepts in the module, and do not reference any concept outside the module. In addition Cuenca Grau *et al* [4] state "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 reuse the entity." This concurs with the definition above.

However, the definition above provides more as it implies that modules are not isolated entities but are related to each other. The notion of inter-module relationships is analogous to the notion of coupling in the Software Engineering paradigm. Highly-coupled ontology modules could be generated via ε-connections[see [3]], and loosely coupled via URI's. The approach presented in this paper does not consider inter-module relations.

Modularising O into different ontology modules, via module extraction, will not mean that each module is disjoint. For example, if A has subclasses B and C then creating a module of A would include all three concepts, but creating a module on B would only include B. Therefore, the module of B is not disjoint from the module of A. However, when constructing an ontology from ontology modules, the modules are likely to be disjoint. This is because the Ontology Engineer is more than likely to pick modules that are about distinct concepts rather than concepts that are closely related, although the domain of the modules is likely to be the same or closely related. It seems somewhat counter-intuitive for an Ontology Engineer to build an ontology from overlapping/intersecting ontology modules. If an Ontology Engineer were to do this then they would be negating the many benefits of having a modular ontology.

It is important to note that an ontology module will probably not be semantically identical to the parent ontology, as the process of extracting an ontology module will result in semantic information contained in the parent ontology not being transferred to the ontology modules. However, it should be possible to regain all of this semantic information by recombining the ontology modules. The impact of this effect will depend on how the process of modularisation is carried out and then reversed. Recombination is not addressed in this paper as a complete partitioning of the ontology is not carried out. The approach presented extracts an ontology module and does not modify the original ontology.

The definition above of an ontology module, in addition, implies that it must be possible to extract modules from the ontology modules themselves, which further increases the opportunity for reuse. This is because an ontology module is self-contained, which in essence means every ontology module is an ontology. However, there may be a point at which an ontology module produced automatically becomes so small that the overhead involved in the reuse step is greater than producing the ontology module manually from scratch. This would only occur if the automatic methods for producing an ontology module are inefficient. As such, in order to support effective information integration, attention needs to be paid to this point to ensure that the methods available for reuse are efficient.

3.1 Ontology Module Requirements

It is now possible to define what requirements are placed upon the ontology modules. The requirements for an ontology module are as follows:

- **Self-contained.** Ontology modules should be a self-contained subset of parent ontology. Given a set of relations the ontology module should be transitively closed with respect to these relations.

- **Concept centred.** The ontology module contains enough information to describe the start concept. Direct superclasses are considered unimportant because they only place the start concept in context. It is assumed that the Ontology Engineer already has a context in mind for the ontology module. In addition including superclasses would increase the chances of the ontology module being equal to the whole ontology.

- **Consistent.** Ontology modules should be consistent. Given a consistent ontology to extract a module from, the module produced should be consistent.

4. METHODOLOGY FOR REUSING AN ONTOLOGY MODULE

The requirements outlined in the previous section allowed the identification of the main steps involved in reusing an ontology; these are summarised in the tentative methodology below. Figure 1 shows our proposed methodology for reusing an ontology module. Although ontology modularisation has been the subject of much interest recently, to the best of our knowledge, nobody has defined a methodology for reusing an ontology module. Thus, Figure 1 is an attempt to draw interest towards this important area. The following explains all the steps involved in the methodology.

Define competency of module Before the module extraction process begins the Ontology Engineer needs to define competency questions for the module; these should state what the Ontology Engineer wants the ontology module to express. The competency questions can be expressed in terms of SPARQL queries. If the competency of the ontology module is not defined in some way then the Ontology Engineer will have no idea whether the ontology module they extract meets their requirements.

Ontology Selection The Ontology Engineer needs to select the relevant ontology that they wish to extract a module from, as the number of ontologies available continues to increase this step will become increasingly important. Sabou *et al.*[19] provide an extensive review on this area of research and also state requirements for the ontology selection process. Whilst ontologies can be discovered by browsing the available repositories or by using a search engine like Swoogle, the major effort in this task will be carrying out the ontology evaluation.

- **Ontology Evaluation** The Ontology Engineer needs to evaluate any ontology that is discovered to see how suitable it is for reuse. There are several criteria that can be applied to evaluate an ontologies suitability for reuse. One criterion could be the ontologies popularity; if an ontology is popular and widely used then it is likely to be correct and of some quality. Another criterion could be to check the ontology against the competency questions the Ontology Engineer defined for the ontology module. If the ontology does not meet the competency questions then the ontology module will most likely not either.

Ontology Translation This step aims to change the representational language used to implement the ontology. This may be necessary to make the ontology compatible with the module extraction process being used. However, the application of this step requires some care to be taken by the Ontology Engineer. Translating an ontology from a more expressive formalism such as OWL-DL to RDFS will cause some ontological features, such as restrictions on the classes, or cardinality constraints to be lost. Conversely, the translation from a less expressive formalism into a more expressive one will cause the Ontology Engineer to perform a further enrichment step that is needed to complement the less expressive model obtained through translation. A complete treatment of the problems associated to ontology translation can be found in [2].

Extract module The ontology module is extracted from the selected ontology. The amount of effort required by the Ontology Engineer in this step will depend on the method of extraction that is being used. In the following section the algorithm we propose for module extraction is presented.

Check Competency The extracted ontology module is checked against the competency questions, in order to see if they are met. One possible way to check whether the competency questions are met is by formulating them as SPARQL queries. If the competency questions are met then the Ontology Engineer can integrate the ontology module into the ontology that they are con-

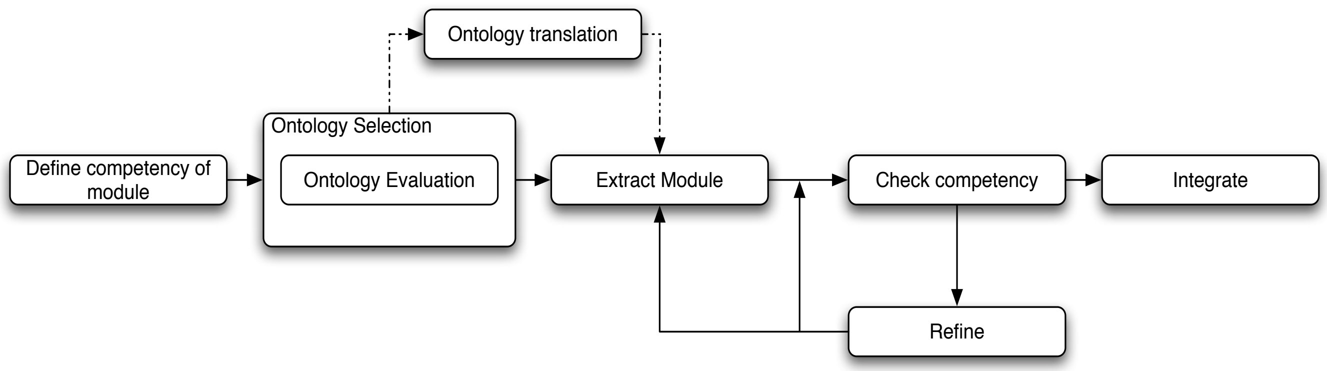

Figure 1: A methodology for reusing an ontology module.

structing. If the competency questions are not met then the Ontology Engineer should refine the ontology module.

Refine When the competency questions are not satisfied then the Ontology Engineer should refine the module in a way that attempts to ensure that the competency questions are met. This refinement could take two possible forms. Firstly, the Ontology Engineer could alter the ontology module, i.e. by adding or removing statements, in order to make it meet the requirements defined by the competency questions. However, under certain circumstances the Ontology Engineer may decide that this would take too much effort and thus decide to run the module extraction process again, or indeed run a different module extraction process. Naturally if the same module extraction process is being used then the Ontology Engineer should ensure that the parameters are fixed differently.

Integrate Once the ontology module satisfies the competency questions it can be integrated into the ontology being constructed.

5. ABSTRACT GRAPH MODEL FOR ONTOLOGY MODULE EXTRACTION

To enable the development of an ontology module extraction method that was as language neutral as possible and fulfilled the requirements stated in Section 3.1, an abstract model was required. The model proposed is an edge-labeled directed graph G, given an alphabet $\sum_E$, is an ordered pair $G = (V, E)$ where:

- V is a finite set of vertices,
- $E \subseteq V \times \sum_E \times V$ is a ternary relation describing the edges (including label). (N.B. E is not symmetric which gives us direction. Therefore to properly capture the definitions of 'disjoint' and 'equivalent' two edges are required.)

Using this abstract model, it is possible to define an ontology module as $G_M = (V_M, E_M)$, where
$V_M \subseteq V \wedge V_M \neq \emptyset$ and $E_M \subseteq E$. This implies that $G_M \subseteq G$.

It is possible to reduce module extraction to the traversal of a graph given a starting point x such that $x \in V_G$. The only exception being that there is no need to traverse '*disjoint*' labeled edges of x in the first iteration. Thus, the module is a graph $G_M = (V_M, E_M)$ where V_M and E_M are the sets of traversed vertices and edges respectively. The minimum number of possible G_M derivable from G should be equal to the number of elements in V. This is because a single module could be generated for each concept.

Algorithm 1 Module Extraction

```
INPUT
  • A directed graph $G = (V, E)$
  • $s$ a starting vertex such that $s \in V_G$
  • Excluded - a container of $E$ not to be followed
  • Visited - a container of $V$ that have been visited
  • ToVisit - a container of $V$ to be visited.
OUTPUT
  • A directed graph $G_M = (V_M, E_M)$
procedure $extractModule$(Vertex $s$)
if $s \notin$ Visited then
  insert $s$ into Visited
  create container $X = \{e \in E | s \times \sum_E \times v\}$
  while $X$ is not empty do
    $y$ =first element of $X$
    if $y \notin$ Excluded then
      $y \cup E_M$
      insert $r$ such that $y = s \times \sum_E \times r$ into ToVisit
    end if
    if ToVisit is not empty then
      $t$ =first element of ToVisit
      remove $t$ from ToVisit
      $extractModule(t)$
    else
      output $G_M$
    end if
  end while
end if
```

Algorithm 1 presents the pseudo-code description of the ontology module extraction method. The complexity of the algorithm is $O(n^2)$. The graph is conditionally traversed to extract the ontology module.

In the case of OWL in the first iteration of the extraction process the disjoint relation is not traversed, but in subsequent iterations it is. This exception is allowed because the user explicitly chooses the concept that the process will start on. If this concept has disjoint concepts the assumption is made that the user is not interested in these concepts. This is fair because disjointedness requires that the concepts have no instances in common. If the user had wished to include the disjoint concepts they should have started the process on their common superclass.

An interesting result of applying this algorithm to generate an ontology module, is that the ontology module produced will be transitively closed with respect to the relations that are traversed. The only caveat is that the ontology being used to obtain the module must be transitively closed with respect to these relations in order to guarantee that the module will also have transitive closure.

It is important to note that there is no upward navigation of the subclass hierarchy from the concept the process was started on. Again this is justified because the user chooses the starting concept. Furthermore, allowing upward navigation of the subclass hierarchy would substantially increase the chance of extracting a module that is equal to the whole ontology.

The abstract graph model means that the module extraction process is independent of the language. For example, the alphabet for OWL-DL is $\sum_E = \{subClassOf, disjointWith, equivalentTo, subPropertyOf, property\}$ and for RDFS it is $\sum_E = \{subClassOf, subPropertyOf, property\}$. Notionally these labels correspond to the primitives of OWL-DL apart from '*property*'. If an edge is labelled '*property*' then it means the starting vertex is the domain and the ending vertex is the range.

Algorithm 1 has been implemented in ModTool, a standalone tool for ontology modularisation. It provides an intuitive graphical user interface (GUI) for extracting ontology modules. It does not produce an exhaustive partition nor does it support the recombination of ontology modules in order to obtain the semantics of the original ontology. It makes use of JENA[8] to allow the ontology being modularised and the ontology module produced to be persistently stored. This is important because dealing with large ontologies in memory is cumbersome. Pellet[9] is also used to check that the modules produced are clean and valid. The GUI is designed to allow the user to select which concept they wish to produce an ontology module about. It is also possible for the user to configure local redirections of any imported ontologies.

5.1 A Walkthrough Example.

Figure 5 shows a simple ontology, represented in the abstract model presented in this paper, about the University domain. From this ontology we shall extract an ontology module about 'Academic Staff', this is the starting concept.

Iteration 1.

'Academic Staff' is added to Visited. 'Academic Staff' is disjoint with 'Admin Staff' so 'Admin Staff' is not added to ToVisit. 'Acadmic Staff' has two subclasses, 'Lecturer' and 'Research Staff', these are added to ToVisit. 'Academic Staff' has no more edges to traverse and is removed from ToVisit; the extraction now continues with 'Lecturer' as the concept of focus.

Iteration 2.

'Lecturer' is added to Visited. 'Lecturer' is the domain of the object property 'supervises'; the range of this property is 'PhD Student', thus 'PhD Student' is added to ToVisit. 'Lecturer' has no more edges to traverse and is removed from ToVisit; the extraction now continues with 'Research Staff' as the concept of focus.

Iteration 3.

'Research Staff' is added to Visited. 'Research Staff' has one subclass 'PhD Student'. 'PhD Student' has already been added to ToVisit, which does not permit duplicate elements, however the edge that describes the subclass relation will be included in the module. 'Research Staff' has no more edges to traverse and is removed from ToVisit; the extraction now continues with 'PhD Student' as the concept of focus.

Iteration 4.

'PhD Student' is added to Visited. 'PhD Student' has no valid edges to traverse. Eventhough 'PhD Student' is the range of an object property, this edge is not traveresed. 'PhD Student' is removed from ToVisit. ToVisit is now empty; the extraction process ends and the ontology module is outputted.

6. EXPERIMENTAL EVALUATION

As a proof of concept preliminary experiments were conducted using the NCI Oncology Ontology[10] and were based on those carried out by Stuckenschmidt & Klein[22]. This allowed for a comparison of results. ModTool was run for every 'partition name' defined by Stuckenschmidt & Klein. For Stuckenschmidt & Klein 's [22] experiments with SUMO, the partition name is derived from the top concept of the respective module. We assume that the same method was used to define the partition names for the NCI Oncology Ontology. Where the partition name did not directly coincide with a concept in the NCI ontology the most similar name was used. There were only 12 cases where this occurred. Full results for the experiments can be found at `www.csc.liv.ac.uk/~pdoran/ experiments/`. The results obtained show that the average difference between ModTool module sizes and those of Stuckenschmidt and Klein was 69, with 40 of the modules being different in size.

It was necessary to compare ModTool's approach with that of Cuenca Grau *et al.*[4]. However, it should be noted that the approach taken by Cuenca Grau *et al.* is biased toward efficiency rather than reuse, and that in fact, the ModTool view is orthogonal to Cuenca Grau *et al.*. ModTool aims to extract ontology modules rather than partitioning. This is due to the Ontology Engineering perspective taken where a user has to find the most suited fragment of an ontology in order to include as much knowledge as is necessary. This enables the conceptualisation to be reused. Whereas the perspective taken by Cuenca Grau and colleagues is that of an ontology maintainer that wants to subdivide a huge ontology into well separated fragments. Thus, the problem of reuse is addressed partially and indirectly. Indeed there

[8]http://jena.sourceforge.net/

[9]http://www.mindswap.org/2003/pellet/index.shtml

[10]http://www.mindswap.org/2003/CancerOntology/nciOncology.owl

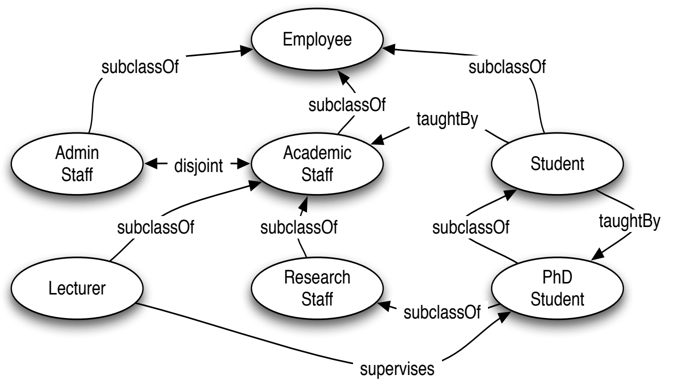

Figure 2: A simple ontology of the University domain.

Ontology	Species	Equivalent	Disjoint	Restriction
AKT-Portal	$\mathcal{ALCHIOF}(\mathcal{D})$		X	X
MindSwappers	$\mathcal{ALCHIF}(\mathcal{D})$		X	
Family	$\mathcal{ALC}$	X	X	X

Table 1: Table showing ontology properties.

is no assurance that a user is interested in a concept that appears in just one fragment. In addition modules cannot have axioms in common but they can share concepts. ModTool instead explores the problem from the perspective of a user that wants to elicitate the knowledge related to a single concept without requiring a complete partition of the ontology. An advantage of such an approach is that a safety check is less constraining since it is no longer required that a whole ontology can be subdivided into disjoint axiom sets, but rather all that is necessary for such a concept is included.

It was felt unfair to compare the ModTool approach with that of Noy and Musen [16] because their approach is oriented to query answering rather than reuse. One drawback of both Stuckenschmidt and Klein, and Noy and Musen is that they do not produce a reusable entity. The output of Stuckenschmidt and Klein partitioning algorithm is not in a Semantic Web language. Noy & Musen only create a view on an ontology which is intended to be used for query answering. It does not allow for the extraction of this view for reuse. Whilst Cuenca Grau and colleagues method does produce reusable entities, their reuse is restricted due to the tight coupling that ε-connections [13] gives them.

In regards to Cuenca Grau and colleagues' method it is not possible to have an in-depth comparison of results due to the size of the partitions produced. Furthermore, the bias of their approach toward efficiency also hinders a more thorough analysis. It is not possible to apply the ε-connections method to the partitions to further reduce their size so that a better comparison can be made. This indicates that under certain circumstances it would be preferable for a user to produce a module using ModTool rather than trying to reuse one of the partitions obtained from using ε-connections. For example, when the partition contains disparate concepts. This is especially the case when one considers the perspective of an Ontology Engineer who is trying to reuse an ontology module, that is likely to need changing in order to conform to their conceptualisation. This would suggest that the ontology modules produced by ModTool are more versatile than the partitions produced by ε-connections. Also, the tight coupling of ε-connections is a disadvantage.

When running the Cuenca Grau and colleagues approach deployed in SWOOP[11] the NCI Oncology ontology was split into 17 partitions. Some of these were large, for example, one partition had 7663 concepts. This compares unfavourably with the modules produced by ModTool whose largest was 2380.

One problem identified with this preliminary experimental comparison was that there is no definitive notion of what a 'good' module is. Plus, the only way to carry out a comparison was on purely subjective terms. Thus, there is a need for a formal objective criteria. We propose that the precision and recall metrics used by Dellschaft & Staab[6] can be applied to modularisation. This requires that a definition of what precision and recall mean to module extraction for reuse be devised.

- **Precision.** All the taxonomical relations that are in the module are also in the parent ontology.

- **Recall.** Everything that is in the parent ontology is in the module.

The metric then calculates the harmonic mean of precision and recall, this is the fMeasure. Recall is particularly important to evaluate the quality of a module, as it provides an objective measure of how much of the original ontology has been retained in the module, thus indirectly measuring how competent the module is. In Uschold and Gruninger[23], the notion of formal competency questions to evaluate the quality of an ontology is introduced. These are used to verify that the ontology contains enough concepts to answer some questions defined by the ontology engineers to limit the scope of the ontology. Analogously, recall reflects how much of the original ontology has been included in the module, thus providing an indirect measure of the scope of the module, or its competency.

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

Ontology	Average Precision	Average Recall	Average fMeasure
Portal	1	0.54	0.68
MindSwappers	1	0.72	0.79
Family	1	0.84	0.9

Table 2: Experimental Results

6.1 Experimental Procedure And Results

Experiments were conducted on three ontologies: the AKT-Portal, MindSwappers and Family ontologies. These ontologies were chosen because of their varying expressiveness, see Table 1. Whilst the Family ontology is the least expressive it is highly interconnected and contains complex restrictions. For example, the Grandfather concept is defined as a father who has a child who is a parent. The ontologies used in the proof of concept, such as the NCI Oncology ontology, were relatively simple and often only formed by taxonomic relations

A module was produced for each concept. The metric was then run to calculate the precision and recall of the module with respect to the original ontology. It is expected that precision will be high because the approach does not make any changes to the concepts that are placed in the module. In addition, intuitively, the recall should vary dependent on how large the module is and where it appears in the taxonomy. For example, large modules at the top of the taxonomy should have a high recall; whilst small modules at the bottom of the taxonomy should have a low recall. There is little expectation that recall will score highly. Indeed it is undesirable to score highly on recall because this indicates that the ontology module is fairly similar to the original ontology, which nullifies the benefits of the module extractions process. Thus, it is expected that the score for recall will be lower than that for precision.

The results are shown in Table 2. They concur with our expectations. Precision was high for all three ontologies; recall was lower, as expected.

The difference in recall between Portal and Mindswappers can largely be attributed to the difference in average depth of the class tree. The average depth of the class tree for Portal is 5.89 and for Mindswappers it is 3.9. The average branching factor and the avergage number of object properties per concept are not significantly different. The average branching factor of Portal is 2.53 and for Mindswappers it is 2.58. Whilst the average number of object properties per concept for Portal is 1.56 and for Mindswappers it is 1.26.

The difference in the avergage depth of the class tree attributes to the difference in recall because of the way the subclass rule works in the extraction process. The greater the depth of the class tree means that as the process gets further down the hierarchy, then less of the hierarchy is placed into the ontology module. Thus, the less in your ontology module the lower the recall.

There is a need for further analysis of the results with regard to recall. This should look at whether a threshold can be defined to assess when an ontology module is competent. The upper threshold would indicate that the module is becoming to general, and the lower threshold would indicate that the module is too specific.

7. CONCLUSIONS AND FUTURE WORK

This paper offers a new approach to generate an ontology module for reuse. The abstract graph model for ontology module extraction allows us to explicitly deal with the different types of relations that are used within RDFS and OWL. Experimental results have shown that ModTool can generate small modules that retain the properties of the original ontology. Conducting usability tests could further validate the approach, in order to recognise if the overhead of taking another's ontology and including it in your own modeling task is manageable, or if it is easier for the Ontology Engineer to model everything from scratch.

The postulation that precision and recall can be used as an objective measure, it is hoped, will allow for different approaches to be compared in a more unbiased manner. This highlights the need for the ontology modularisation community as a whole to decide upon what a 'good' module is. It would also be useful for a single ontology or a suite of ontologies to be selected as a benchmark. These are important steps as they will allow different approaches to be compared more objectively.

A flaw in the implementation of ModTool is that if concepts in the ontology module inherit object properties from concepts that are not in the module then these object properties are not included. This is not a trivial problem to solve because deciding whether or not to change the domain and range of an object property is not simple. For example, class A has subclass B and object property P has domain A and range C. If the user starts the module extraction process on B then do we need to include A? If A is included then the module size is likely to increase because upwards navigation is introduced, and in certain cases the only module one may be able to produce will be equal to the whole ontology. Or is it possible to change the domain from A to B? If the domain is changed does the range need to be changed and how can this be decided? This is not a major flaw as the user is able to add these properties by hand later if they are required; but an automatic solution to this problem is a current focus of research.

In the future, the ModTool approach will also be applied to the other goals of modularisation. For example, the ModTool approach could allow for quicker and more relevant information retrieval by narrowing the scope of the query. In addition, mechanisms to handle evolution and maintenance need to be added to ModTool. Although strict enforcement of evolution and maintenance is unlikely with the approach taken, Ontology Engineers still need to be made aware when the ontology they obtained their ontology module from has been changed. Additionally research needs to be carried out to see if ModTool can reduce the cognitive complexity for Ontological Engineers allowing them to develop a more accurate understanding of the ontologies they are reusing. A subject of current research is investigating the effect of using more than one concept to instantiate the module extraction process. This could be useful if an Ontology Engineer cannot find one specific concept in the ontology that fits their needs, but rather needs to select several to obtain a module that fits their requirements.

8. ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) and Knowledge Web(KW).

9. REFERENCES

[1] A. Bernaras, I. Laresgoiti, and J. Corera. Building and reusing ontologies for electrical network applications. In W. Wahlster, editor, *Proc. of the 12th ECAI*, pages 298–302, 1996.

[2] O. Corcho. *A Layered Declarative Approach to Ontology Translation with Knowledge Preservation (Frontiers in Artificial Intelligence and Applications)*. IOS Press, US, 2005.

[3] B. Cuenca Grau, B. Parsia, and E. Sirin. Combining owl ontologies using e-connections. *Journal Of Web Semantics*, 4(1):1–42, 2005.

[4] B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularizing owl ontologies. In *Proceedings of the KCAP-2005 Workshop on Ontology Management*, Banff, Canada, 2005.

[5] M. d'Aquin, M. Sabou, and E. Motta. Modularization: a key for the dynamic selection of relevant knowledge components. In *First International Workshop on Modular Ontologies, ISWC 2006, First International Workshop on Modular Ontologies, ISWC 2006*, Athens, Georgia, USA., 2006.

[6] K. Dellschaft and S. Staab. On how to perform a gold standard based evaluation of ontology learning. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 228–241. Springer, 2006.

[7] J. D. (Editor), D. F. (Editor), and F. van Harmelen (Editor). *Towards the Semantic Web: Ontology-driven Knowledge Management.* John Wiley and Sons Ltd, UK, 2002.

[8] N. Fridman Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of protege-2000: Combining interoperability and flexibility. In R. Dieng, editor, *Proceedings of the 12th EKAW Conference*, volume LNAI 1937, pages 17–32, Berlin, 2000. Springer Verlag.

[9] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? a case for conservative extensions in description logics. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 187–197. AAAI Press, 2006.

[10] A. Gómez-Pérez, M. Fernández-Lopez, and et al. *Ontological Engineering.* Springer, London, 2003.

[11] A. Gómez-Pérez and D. Rojas-Amaya. Ontological reengineering for reuse. In *Knowledge Acquisition, Modeling and Management: 11th European Workshop, EKAW '99, Dagstuhl Castle, Germany, May 1999. Proceedings*, volume 1621 of *Lecture Notes in Computer Science*, page 139. Springer Berlin, 1999.

[12] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. A logical framework for modularity of ontologies. In Veloso [24], pages 298–303.

[13] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of abstract description systems. *Artificial Intelligence*, 156(1):1–73, 2004.

[14] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In Veloso [24], pages 453–458.

[15] N. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics (SMI), Department of Medicine, Stanford University School of Medicine, 2001.

[16] N. F. Noy and M. A. Musen. Specifying ontology views by traversal. In *International Semantic Web Conference*, pages 713–725, 2004.

[17] A. Rector, A. Napoli, G. Stamou, G. Stoilos, H. Wolger, J. Pan, M. D'Aquin, S. Spaccapietra, and V. Tzouvaras. Report on modularization of ontologies. Technical report, Knowledge Web Deliverable D2.1.3.1, 2005.

[18] A. L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture*, pages 121–128, New York, NY, USA, 2003. ACM Press.

[19] M. Sabou, V. Lopez, E. Motta, and V. Uren. Ontology selection: Ontology evaluation on the real semantic web. In *Proceedings of the EON'2006 Workshop, "Evaluation of Ontologies on the Web", held in conjunction with WWW'2006*, 2006.

[20] J. Seidenberg and A. Rector. Web ontology segmentation: analysis, classification and use. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 13–22, New York, NY, USA, 2006. ACM Press.

[21] H. Stuckenschmidt and M. Klein. Integrity and change in modular ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'03*, Acapulco, Mexico, 2003.

[22] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In *Proceedings of the 3rd International Semantic Web Conference*, Hiroshima, Japan, 2004.

[23] M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.

[24] M. M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.