Ontology reuse via ontology modularisation

Paul Doran
Department of Computer Science, University of Liverpool,
Liverpool, L69 3BF, UK
pdoran@csc.liv.ac.uk

1 Motivation

The realisation of the Semantic Web depends on a number of factors, one of which is the ability to reuse ontologies [10]. Ontology construction is deemed to be a time consuming and labour intensive task. Therefore it heavily relies on the possibility of reusing existing ontologies. Nonetheless, ontologies and ontology libraries, for instance the DAML ontology library[1] are becoming available, together with the first search engines for the retrieval of web ontologies, such as Swoogle[2] and Ontosearch[3]. However, from an Ontological Engineering perspective it can be hard to reuse ontologies, especially when the ontologies being reused are large. Currently there are two major ways of reusing an ontology. Ontology editors such as Protégé [5] 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). The web ontology language OWL[1] offers the possibility to *import* an OWL ontology by means of the `<owl:imports>` statement. In both cases, these include the whole ontology. However, especially when large ontologies are reused, ontology engineers might be interested only in a portion of the original ontologies. For example, an engineer is only interested in concepts about and relating to diabetes, but they had to import the whole ontology in order to get what was required.

Reusing only part of an ontology has been the starting point for this PhD research that aims to investigate the wider notion of *ontology modularisation*. Ontology modularisation is a mechanism that allows the Ontological Engineer to extract a sub-set, an *ontology module*, of the original ontology. However, the goals of modularisation are not only restricted to ontology reuse, but also include scalability for information retrieval, scalability for evolution and maintenance, complexity management, understandability, personalization and reuse [8]. The focus of this paper is on ontology modularisation for reuse. This is the first problem tackled by this PhD research, begun in September 2005. In particular, with this effort the aim is to answer the following research question:

> Is it possible to separate only part of an ontology, in a way such that a new ontology is generated, centered around a specific concept, and that is correct and self contained, and that can be reused instead of the original one?

Thus the aim is to reduce the size of the ontology being imported to the part that is deemed relevant for the new application. Furthermore, the Engineer can make modifications to the ontology module to tailor the representation to his application. In order to test this approach to modularisation, ModTool was developed. This is a tool for producing ontology modules from OWL ontologies.

2 Related work

This section aims to place the work done with ModTool in the context of the current research direction and outline arguments for why it was necessary. In order to do this the approach taken will be compared and contrasted with that taken by others.

Firstly, it is necessary to briefly outline the approach taken by the methods that ModTool will be compared and contrasted with. The method

[1]http://www.daml.org/ontologies/
[2]http://swoogle.umbc.edu/
[3]http://www.ontosearch.org/

taken by [12] is that of partitioning a large concept hierarchy. The ontology being partitioned is effectively turned into a semantic net. [7] creates a 'view' on the ontology. This is produced by traversal of relations that are specified by the user. [4] however, take a description logic partitioning based approach to modularisation.

One drawback of [12] and [7] is that they do not produce a reusable entity. The output of [12]s partitioning algorithm is not in a Semantic Web language. [7] 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 [4] does produce reusable entities, their reuse is restricted due to the tight coupling that ε-connections gives them. Thus, the intention was that ModTool should produce highly reusable entities in the form of ontology modules.

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 [12] and [4] are automatic and require minimal user involvement. This obviously makes them fairly simple to use, but there is little control over the results. [7] requires user involvement and the results of the method can therefore be tailored by the user to their exact requirements. As a result, when constructing ModTool the requirement that the process was user-led was decided upon due to the ontological engineering perspective. In addition, there was a need for the ontology module produced to be user customisable. It was hoped that this would make the ontology modules highly reusable because the Engineer can modify the module to conform to their conceptualisation of the domain.

[12] 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." This is a drawback of their approach because considering what the relations 'mean' is important in determining modules. This is why the rules applied by ModTool independently consider the different types of relations. Whilst [7] allows the user to state what relations are to be considered their method is based on RDFS and therefore it is hard to see how it could be successfully deployed on OWL ontologies. Consequently, ModTool was tailored specifically to OWL. Whilst [4]'s method applies to OWL and also considers, at some level, the semantics. ε-connections have limited expressiveness of the connections made between partitions. ε-connections cannot express subclass relations or sub-properties. As such, when developing ModTool an emphasis was placed upon ensuring that there were as few as constraints as possible on the output produced. This allows the user to tailor it to their specific individual requirements.

Thus, the ModTool approach makes several contributions. The first being that it produces highly reusable entities in the form of ontology modules. The user of ModTool is also able to select which concept the process begins on. This allows the user to generate the most relevant ontology module for their application.

3 Defining An Ontology Module

Before proceeding to describe the approach to ontology modularisation currently being investigated, it is necessary to formalise and define what an ontology module is. 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 [12]. Therefore, what is an ontology module? A possible definition for 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 relationship to other ontology modules.*

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 ontology modularisation 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.

The validity of this definition needs to be assessed in relation to the literature. [3] state "a module should contain information about a self-contained subtopic." This concurs with [11] 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 [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 is in agreement with the definition above.

Most ontologies are developed under the open world assumption, thus a reference to a concept outside the ontology is permissible, but in order to obtain self-contained (i.e. no references to concepts outside the ontology module) ontology modules the closed world assumption is required. This means that a reference outside of the ontology evaluates to false rather than unknown. This requires that the open world assumption be relaxed in order to verify whether ontology modules are self-contained.

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.

A formal representation (based on [6]) of the definition is, $O = (C, R)$ where O is an ontology where $C = \{c_1 \ldots c_n\}$ is a set of concepts and $R = \{R_1(a\ ,\ b) \ldots R_n\ (a\ ,\ b)\}$ is a set of relations (including functions). Thus, it is possible to define an ontology module as $O_M = (C_M\ ,\ R_M)$, where $C_M \subseteq C \wedge C_M \neq \Theta$ and $R_M \subseteq R$. This implies that $O_M \subseteq O$. Furthermore, the minimum number of possible O_M derivable from O should be the number of elements in $C + 1$, thus, every ontology will have at least $n+1$ ontology modules, if n equals the number of concepts in the ontology. The only exception to this being an ontology with only one concept. However, an ontology with only one concept cannot be modularised as it is atomic.

Modularising O into different ontology modules 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.

It is important to note that an ontology module will probably not be semantically identical to the parent ontology, as the process of defining 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 most, if not 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.

The definition above of an ontology module, in addition, implies that it must be possible to modularise the ontology modules themselves, which further increases the opportunity for reuse. This is because an ontology module is complete, which in essence means every ontology module is an ontology. However, there must be a point at which an ontology module produced automatically becomes so small that the overhead in reusing it is greater than producing the ontology module manually from scratch.

4 ModTool approach to modularisation

ModTool has been implemented as a standalone tool. It provides an intuitive graphical user interface (GUI) for generating ontology modules. ModTool recursively applies a set of rules to generate an ontology module. It makes use of

JENA[4] 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[5] is also used to check that the modules produced are clean and valid.

Algorithm 4.1: Ontology Module Generation()

```
procedure BEGIN($Conceptstart, Ontologyparent$)
 ADD start to conceptsToWorkOn
 READ parent from file
 $i = 0$
 MODULARISE()

procedure MODULARISE()
 SET activeConcept to first element of
 conceptsToWorkOn
 ADD activeConcept to module
 if $i = 0$
   then do nothing
   else
     for each Disjoint d activeConcept has
     in parent
     if d is not in module
       then
         ADD d to module
         ADD d to conceptsToWorkOn
 for each Equivalent e activeConcept has
 in parent
   if e is not in module
     then
       ADD e to module
       ADD e to conceptsToWorkOn
 for each DirectSubClass s activeConcept has
 in parent
   if s is not in module
     then
       ADD s to module
       ADD s to conceptsToWorkOn
 for each ObjectProperty o that activeConcept
 is the domain of in parent
   if o is not in module
     then
       ADD o to module
       ADD o to conceptsToWorkOn
 for each DatatypeProperty p that
 activeConcept is the domain of in parent
   if p is not in module
     then
       ADD p to module
 REMOVE activeConcept from
 conceptsToWorkOn
 if conceptsToWorkOn is empty
   then
     check module is clean and valid
     PRINT module to file
   else
     $i++$
     MODULARISE()
```

Figure 1. ModTool's algorithm

The GUI is designed to allow the user to select which concept they wish to produce an ontology module about. The user also has to configure the database settings. It is also possible for the user to configure local redirections of any imported ontologies. The display area to the right of the GUI allows the user to track what is happening within the ontology modularisation process. Figure 1 presents the pseudo-code description of the method used by ModTool to generate ontology modules. Essentially a series of rules are applied. There is a rule for each type of relation that exists in an OWL ontology. The application of rules adds concepts and relations to the module. Then when there are no more concepts that the rules can be applied the module is outputted as an OWL file.

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

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

The interesting result of applying these rules to generate an ontology module, is that the ontology module produced will be transitively closed. It is possible to abstract all the rules to a common form: *IF* `a is domain of R`
THEN `add range b of R to module`.

Thus, recursive application of this results in transitive closure because all relations are traversed until they reach their leaf node. Thus, the issue of relaxing the open world assumption has thereby been avoided. The only exception to this in Algorithm 4.1 is the disjoint rule in the first iteration. 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 defines 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.

The only caveat is that the ontology being used to obtain the module must be transitively closed in order to guarantee that the module will also have transitive closure. However, in certain cases it may be possible to obtain a module that has transitive closure when the ontology it was derived from did not.

5 Evaluation

Evaluation is an integral part of the research methodology followed, and this is outlined below. To start the existing literature on Ontology Engineering was looked at, paying particular attention to the various methodologies for extracting ontology modules. An attempt was then made to outline the main features that an ontology module should have, and these were formalised in the definition given in Section 3. Next an algorithm that built modules characterised by this definition was designed. This algorithm was implemented in ModTool . Tests were then conducted to test whether the algorithm provided a comparable performance in terms of module size and correctness with respect to the methods presented in Section 2. In particular here the analysis of ModTool 's performance with respect to the method used by Stuckenschmidt and Klein.

The experiments conducted used the NCI Oncology Ontology[6] and were based on those carried out by [12]. This allowed for a direct comparison of results. ModTool was run for every 'partition name' defined by [12]. 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.

5.1 Results

Full results for the experiments can be found at www.csc.liv.ac.uk/~pdoran/experiments/. The results obtained by ModTool were favourable. The average difference between ModTool 's module sizes and those of [12] was 69, with 40% of the modules being different in size.

5.2 Discussion

There is the need to elucidate upon some of the results to further highlight the differences with the ModTool approach. These comparisons can be seen in Figures,2 and 3. The solid black lines denote the ModTool modules and the dashed lines denote [12]'s partitions.

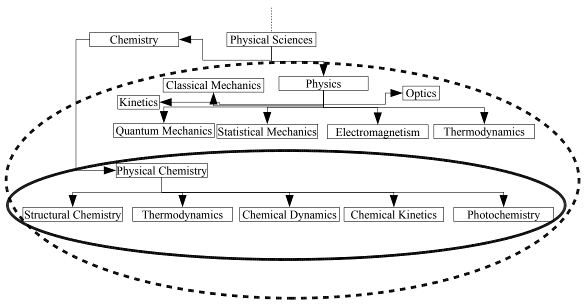

Figure 2. Comparison Of 'Physics & Physical Chemistry' Modules.

Figure 2 shows the 'Physics & Physical Chemistry' module. The module produced by ModTool is a 'Physical Chemistry' module rather than a 'Physics, Physical Chemistry' module because there is no 'Physics, Physical Chemistry' concept. [12] produce the 'Physics, Physical Chemistry' partition by combining two distinct sections of the ontology. The ModTool would produce 'Physical Chemistry' and 'Physics' module separately. This raises the question as to whether it is possible to pass more than one concept into ModTool for the modularisation process to operate upon.

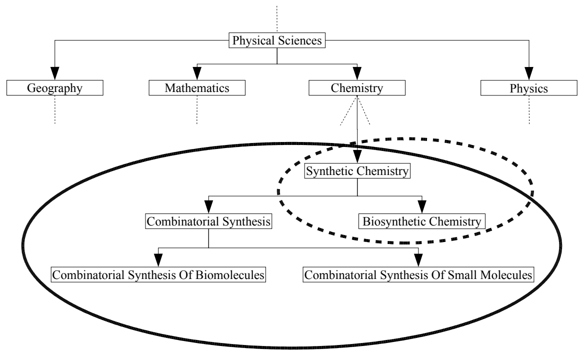

Figure 3. Comparison Of 'Synthesis Chemistry' Modules.

The 'Synthesis Chemistry' module is illustrated in Figure 3. [12]s partition only includes one child of the 'Synthesis Chemistry' concept, this is because there partitioning algorithm also produced a 'Combinatorial Synthesis' partition. This highlights the fact that the partitioning algorithm makes implicit assumptions; this being that the partitions will not be used as stand alone entities. Whereas, the module produced by ModTool includes both the children of the 'Synthesis Chemistry' concept. Thus, it could be argued that the ModTool module is a better representation for a module about Synthesis Chemistry.

6 Conclusions and future work

ModTool offers a new approach to generate an ontology module for reuse, either as it is or with modifications. The approach explicitly deals with the different types of relations that are used within OWL. Furthermore the experiments conducted illustrate that ModTool compares favourably with the other methods for modularisation.

The one flaw to the approach taken by ModTool is that if concepts in the ontology module inherit object properties from concepts that are not in

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

the module then these object properties are not included. This is not a major flaw as the user is able to add these properties by hand if they are required; but an automatic solution to this problem is a current focus of research. Also, a subject of current research is investigating the effect of using more than one concept to instantiate the modularisation process.

In the future, the ModTool approach will also be applied to the other goals of modularisation. For example, do the modules produced allow for quicker and more relevant information retrieval? What mechanisms can be built into ModTool in order to handle evolution and maintenance? Could ModTool reduce the complexity for Ontological Engineers allowing them to develop a more accurate understanding? In addition, there is a need to asses the ModTool approach with [2] because ModTool should be able to modularise contexts as well. Lastly, ModTool needs to be investigated in terms of distributed description logics (see [9]).

References

[1] OWL, web ontology language. `http://www.w3.org/TR/2003/CR-owl-features-20030818/`

[2] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt, 'Contextualizing ontologies', *Journal Of Web Semantics*, **1**(4), (2004).

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

[4] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur, 'Modularizing owl ontologies', in *Proceedings of the KCAP-2005 Workshop on Ontology Management*, Banff, Canada, (2005).

[5] N. Fridman Noy, R. W. Fergerson, and M. A. Musen, 'The knowledge model of protege-2000: Combining interoperability and flexibility', in *Proceedings of the 12th EKAW Conference*, ed., R. Dieng, volume LNAI 1937, pp. 17–32, Berlin, (2000). Springer Verlag.

[6] T.R. Gruber, 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, **5**(2), 199–220, (1993).

[7] Natalya Fridman Noy and Mark A. Musen, 'Specifying ontology views by traversal.', in *International Semantic Web Conference*, pp. 713–725, (2004).

[8] 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).

[9] L. Serafini, A. Borgida, and A. Tamilin, 'Aspects of distributed and modular ontology reasoning', in *Proceedings of IJCAI'05*, pp. 570–575, (2005).

[10] S. Staab, A. Gomez-Perez, W. Daelemana, M.-L. Reinberger, and N.F. Noy, 'Why evaluate ontology technologies? because it works!', *IEEE Intelligent Systems*, **19**(4), 74–81, (2004).

[11] 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).

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