Towards a Parametric Ontology Modularization Framework Based on Graph Transformation

Mathieu d'Aquin[1], Paul Doran[2], Enrico Motta[1], and Valentina Tamma[2]

[1] Knowledge Media Institute (KMi), The Open University, Milton Keynes, UK
m.daquin@open.ac.uk
[2] Department of Computer Science, University of Liverpool , UK
pdoran@csc.liv.ac.uk

Abstract. Several tools and techniques have been proposed recently for extracting modules from ontologies. Being focused on some particular application scenario, modularization tools generally rely on their own definitions and intuitions about modularity. In this paper, it is proposed that these different techniques be expressed under a common framework. As most of the existing approaches rely on the traversal of the graph underlying the ontology to harvest relevant entities and axioms, the framework chosen is based on graph transformation. An abstract model for representing ontologies as attributed graphs is described, along with the reformulation of existing module extraction techniques as graph transformation rules. The ultimate goal of this work is to build a parametric modularization tool, enabling application developers to easily compare, apply and combine different approaches for module extraction, or even create entirely new techniques.
Keywords: ontology module extraction, graph transformation, parametric modularization

1 Introduction

Recently, the idea of modularization has gained important attention from the Semantic Web community, as tools and methods for developing and managing large ontologies are more and more required. This notion of modularization, or modularity, comes from software engineering where it relates to the design of software made of well defined components that can be managed and reused independently. However, as the definition for a good software module is already vague [1], there is no clear agreement on the criteria for decomposing an ontology into modules. Many different approaches have been proposed, in particular for extracting significant modules from existing, and potentially large scale ontologies (see e.g. [2–6]). As shown in [7], these techniques rely on their own assumption about modularity, based on the particular application scenario they are focusing on (reasoning, visualization, evolution, ontology reuse, etc.) This heterogeneity of approaches, aims, and implementations hampers the comparison and reuse of existing tools. A framework is required to facilitate the selection and combination of the appropriate techniques for particular application scenarios.

While they are presented and implemented in different ways, resulting in different modules, techniques for ontology module extraction are generally based on similar principles: they rely on the traversal of the graph structure underlying the ontology to gather entities and axioms considered as relevant for the module. Each technique relies on its own assumptions, its own rules, about which relations should be traversed and which entities and axioms should be included. Based on this observation, it is believed that a graph transformation model [8] can be used as a basis for a common framework for ontology module extraction. In such a framework, a modularization technique would be expressed as a set of transformation rules, applied upon a graph representation of the source ontology. The interest in designing this framework is twofold:

– Providing a common, well founded comparison framework for ontology modularization techniques helps in better understanding the intuitions on which they are based, their relative strengths and limitations, as well as the situations in which they are relevant or not.
– On the basis of this framework, a "parametric modularization tool" can be built, in which different approaches can be exploited, combined and even created by the user, to obtain the modularization that matches the requirements of his application. The *parameters* in that case relate to the set of graph transformation rules, that can be visualized and edited in a simple, "graphical" way.

The proposed framework is based on the structural representation of ontologies in the form of graphs. Another interesting outcome of this work is that it provides a basis for studying the limitations of the considered structural (syntactic) techniques in taking into account the underlying semantics of ontologies. In particular, we expect difficulties in the integration of approaches relying on reasoning (like [4] and [5]).

In this paper, we provide the initial step towards such a framework. Section 2 presents a brief overview of ontology module extraction. Section 3 investigates a model for representing ontologies as graphs and extraction operations as transformation rules. The intention is to show the feasibility and the interest of such an approach by reformulating several existing module extraction techniques as graph transformation rules. This is done in Section 4. This leads to a discussion in Section 5 on the implementation of a tool for parametric modularization based on the framework presented in this paper. Finally, Section 6 presents the conclusions and future work.

2 Ontology Module Extraction

This paper only considers techniques for extracting one ontology module from an existing ontology, what we call *ontology module extraction* techniques. Descriptions of other types of modularization techniques, including ontology partitioning and query based view extraction can be found in [7] and [3].

An ontology is defined as a set of *axioms* (subclass, equivalence, disjointness relations, etc.) The *vocabulary* of an ontology O, also called its signature $Sig(O)$, is the set of entity names that are employed in the axioms of O.

The task of module extraction consists in reducing an ontology O to the sub-part, the module, that covers a particular sub-vocabulary. It has been called segmentation in [3] and traversal view extraction in [2]. More precisely, given an ontology O and a set $SV \subseteq Sig(O)$ of terms from the ontology, a module extraction mechanism returns a module M_{SV}, supposed to be the relevant part of O that covers the sub-vocabulary SV ($Sig(M_{SV}) \supseteq SV$). Techniques for module extraction often rely on the so-called *traversal approach*: starting from the elements of the input sub-vocabulary, relations in the ontology are recursively "traversed" to gather related elements to be included in the module. The relations between these entities are also included to build a self-contained ontology module, taking the form of an ontology (i.e. a set of axioms).

Several tools implementing more-or-less sophisticated techniques have been recently proposed, generally focusing on a particular use-case for modularization. For instance, in [6] Doran *et al* discuss a module extraction approach dedicated to ontology reuse, and [2] is a module extraction tool directly integrated in an ontology development environment (Protégé), where the extracted module is integrated into the ontology currently being edited. Other approaches are involved in specific application scenarios like the selection of relevant knowledge components from online ontologies [4]. Finally, modularization is often viewed as a way to improve the scalability of reasoning mechanisms when dealing with large ontologies. Corresponding techniques have been developed for example in [3] and [5].

All these techniques are detailed in Section 4, where they are reformulated within our common, general framework.

3 A Graph Transformation Model for Modularization

The techniques briefly mentioned in the previous section are intended to be used in different application scenarios. As such, they rely on different intuitions about ontology modularity and, therefore, generally give different results [7]. However, these techniques are all based on the same principles: starting from a sub-set of the named entities of the original ontologies, they traverse the graph underlying the ontology to harvest relevant entities to be included. In this section, we build a graph transformation model, composed of a graph model for ontologies and a rule model for extraction, in order to provide a common framework in which these techniques can be reformulated.

3.1 Representing Ontologies as Attributed Graphs

We chose to rely on directed, attributed graphs, as it is a powerful enough model to represent ontologies written in RDF-S, OWL, or DAML+OIL. The underlying RDF graph[3] model of these formalisms does not contain the features required by the framework presented. In addition, attributed graphs are the model implemented in the AGG library[4] for graph transformation, therefore providing an adequate basis for implementation. Below, we employ a simplified definition for attributed graphs. Details about attributed graphs, their definition and transformation, can be found for example in [8, 9].

Attributed graphs. An *attributed graph* G is a pair (N_G, E_G), where N_G is a set of attributed nodes and E_G is a set of attributed edges. An *attributed node* $n = \langle T_n, AV_n \rangle$ has a type T_n and a set of attribute values AV_n. An *attributed edge* $e = \langle T_e, AV_e, O_e, D_e \rangle$ has a type T_e, a set of attribute values AV_e, an origin node O_e and a destination node D_e. An *attribute value* is a pair (a, v) associating a value v to an attribute a.

Ontologies as attributed graphs. Ontology representation languages are mostly based on the RDF model, that is itself based on graphs. In that sense, representing ontologies as attributed graphs is quite straightforward. We consider four types of nodes: *Class*, *Property*, *Individual* and *Literals*. Edge types correspond to properties used to relate different entities. For example, *subClassOf* is a type of edge that can be used to link nodes of the type *Class*.

In the case of a named entity (i.e. the entity is associated to an identifier, a URI), this name is used as a value for the attribute **name** in the corresponding node. For example, if an ontology contains a class `C`, it would be represented as a node of the type *Class*, with the attribute value **name**=C.

In the case of OWL or DAML+OIL, classes (and to some extent properties) can be represented as expressions, using language constructs and other classes or properties. The employed construct is represented in the class node as a value for an attribute named **const** and special edges are employed to link the corresponding node to the nodes of other entities used in its definition. For example, we use *op*1 and *op*2 for the operands of a union ($\mathtt{A} \sqcup \mathtt{B}$) or an intersection ($\mathtt{A} \sqcap \mathtt{B}$), as well as p and $someValuesFrom$ for an existential restriction ($\exists \mathtt{p.C}$).

A simple example. Following the principles described above, Figure 1 shows the representation as an attributed graph of the expression $PersonWithDogAndCat \equiv Person \sqcap \exists hasPet.Dog \sqcap \exists hasPet.Cat$[5]

3.2 Graph Transformation Rules for Ontology Modularization

Most of the module extraction techniques are based, explicitly or implicitly, on the idea of traversing the graph underlying the source ontology to collect entities or axioms to be included in the module. The main difference between

[3] `http://www.w3.org/TR/rdf-mt/`

[4] `http://tfs.cs.tu-berlin.de/agg/index.html`

[5] in the description logic syntax. Taken from `http://owl.man.ac.uk/tutorial/twopets.rdf`

$$\begin{tikzcd}
\text{N1:}Class\ (\textbf{name}{=}PersonWithDogAndCat) \arrow[d, "\equiv"] & & & \\
\text{N2:}Class\ (\textbf{const}{=}\sqcap) \arrow[rr, "op1"] \arrow[dr, "op2"] & & \text{N3:}Class\ (\textbf{name}{=}Person) & \\
& \text{N4:}Class\ (\textbf{const}{=}\sqcap) \arrow[r, "op1"] \arrow[d, "op2"] & \text{N5:}Class\ (\textbf{const}{=}\exists) \arrow[dd, "p"] \arrow[dr, "someValuesFrom"] & \\
& \text{N6:}Class\ (\textbf{const}{=}\exists) \arrow[dr, "p"] \arrow[d, "someValuesFrom"] & & \text{N7:}Class\ (\textbf{name}{=}Dog) \\
& \text{N9:}Class\ (\textbf{name}{=}Cat) & \text{N8:}Property\ (\textbf{name}{=}hasPet) &
\end{tikzcd}$$

Fig. 1. Attributed graph representation of the expression $PersonWithDogAndCat \equiv Person \sqcap \exists hasPet.Dog \sqcap \exists hasPet.Cat$

these techniques is the rules they apply to decide wether or not an element has to be included in the module. Therefore, based on the attributed graph model for ontologies, we believe that module extraction techniques can be formulated as graph transformation rules.

Graph transformation rules on attributed graphs. An *attributed graph transformation rule* $R = \langle LHS_R, RHS_R, Map_R \rangle$ is composed of an attributed graph LHS_R representing the left hand side of the rule (the *premise*), of an attributed graph RHS_R representing the right hand side of the rule (the *transformation*), and of a set of mappings between LHS_R and RHS_R. *Mappings* are either pairs of nodes (n_1, n_2) or pairs of edges (e_1, e_2), indicating a correspondence between these two elements in two different graphs. For the sake of simplicity, we consider nodes having the same label and edges of the same type between corresponding pairs of nodes to be implicitly mapped.

Transformation rules for module extraction. The role of rules in the case of module extraction is to decide which entities and axioms (relations) have to be included in the extracted module. To represent included elements, a particular boolean attribute is used, **inc**, that can be applied to both nodes and edges. The premiss of a modularization rule generally contains one or more elements having **inc**=$true$. The right hand side of such a rule generally contains additional elements marked as included (**inc**=$true$).[6]

Module extraction techniques take as an input a sub-vocabulary of the ontology. This sub-vocabulary represents the starting point for the application of the transformation rules. The first step of the process is therefore to mark the nodes corresponding to the elements of this sub-vocabulary as included. Then,

[6] To simplify the notation, we will consider the presence of the **inc** attribute as representing **inc**=$true$. In case there is no indication about **inc**, it implicitly means **inc**=$false$.

the transformation rules corresponding to the modularization technique are applied (in random order[7]) until no transformation is applicable. Applying a rule corresponds to matching the left hand side of the rule into a subgraph of the graph representing the source ontology (i.e. finding an isomorphic subgraph) and replacing this subgraph by the one in the right hand side of the rule, taking into account the mappings. Existing graph transformation engines (such as AGG) implement this process and can be used for this purpose.

A simple example: the extraction of a subtree of the class hirarchy. A very simple modularization technique consists in extracting a sub-tree of the class hierarchy of the ontology, specifying (as the input sub-vocabulary) the root class of this sub-tree. The following transformation rule implements this technique:

Premiss	**Transformation**
$\begin{tikzcd} \texttt{C}_1{:}\mathit{Class}\,(\textbf{inc}) \\ \texttt{C}_2{:}\mathit{Class} \arrow[u, "\sqsubseteq"] \end{tikzcd}$	$\begin{tikzcd} \texttt{C}_1{:}\mathit{Class}\,(\textbf{inc}) \\ \texttt{C}_2{:}\mathit{Class}\,(\textbf{inc}) \arrow[u, "\sqsubseteq\,(\textbf{inc})"'] \end{tikzcd}$

Once the input root class is marked as included, the rule is applied recursively until the leaves of the hierarchy are reached, and marked as included. Therefore, at the end of the transformation process, all the entities (nodes) and axioms (edges) representing the considered subtree of the ontology class hierarchy are included in the module.

4 Implementing Modularization Techniques as Graph Transformation Rules

Our goal is to provide a common framework for module extraction techniques in which a common mechanism, graph transformation, would be used and parametrized by specialized transformation rules. In order to show the feasibility and the relevance of such an approach, we detail hereafter the reformulation of several techniques (from very simple ones, to more sophisticated) in our graph transformation model for ontologies. Note that, in addition to providing the basis for a tool for parametric, customisable module extraction, this also gives the opportunity to better understand the intuitions, or assumptions, on which the considered techniques rely, and to compare them on a common basis.

4.1 Galen Segmentation Service

The technique described in [3] has been developed specifically for the Galen ontology, but the core of the technique is generic and can be applied to any ontology.

[7] This assumes that the rules are commutative.

It takes one or several classes of the ontology as an input, and applies the basic idea that anything that participates (even indirectly) to the description (and so the definition) of an included class has to be included. Therefore the (simplified) reformulation of this technique in our graph transformation framework is straightforward:

Premiss	**Transformation**
$\begin{tikzcd} \mathtt{N}_2{:}* \\ \mathtt{N}_1{:}*\ (\textbf{inc}) \arrow[u, "*"'] \end{tikzcd}$	$\begin{tikzcd} \mathtt{N}_2{:}*\ (\textbf{inc}) \\ \mathtt{N}_1{:}*\ (\textbf{inc}) \arrow[u, "*\ (\textbf{inc})"'] \end{tikzcd}$

A node or an edge marked with the type $*$ can be matched with a node or an edge of any possible type in the graph model. Note that, according to this rule, all the super-classes of included classes are included, as well as any definition associated to included classes or properties. Individuals are not included.

4.2 For Knowledge Reuse

The technique described in [6] is focused on ontology module extraction for aiding an Ontology Engineer in reusing an ontology module. It takes a single class as input and extracts a module about this class. This approach is independent of the language in which the ontology is expressed. The traversal carried out for extraction is done conditionally, with the conditions changing to suit the language that the ontology is expressed in. The approach it relies on is that, in most cases, elements that (directly or indirectly) make reference to the initial class should be included. In that sense, as it can be seen in the following rule, this technique can be considered as the inverse of the Galen segmentation service.

Premiss	**Transformation**
$\begin{tikzcd} \mathtt{N}_2{:}*\ (\textbf{inc}) \\ \mathtt{N}_1{:}* \arrow[u, "*"'] \end{tikzcd}$	$\begin{tikzcd} \mathtt{N}_2{:}*\ (\textbf{inc}) \\ \mathtt{N}_1{:}*\ (\textbf{inc}) \arrow[u, "*\ (\textbf{inc})"'] \end{tikzcd}$

It is worth noticing that, using this technique, all the sub-classes of the input classes, as well as any entity using this class or other included entities in its definition, will be included.

The above rule is very simple and shows that this module extraction technique can be easily represented using graph transformation rules. However, it does not completely translate the corresponding technique, as [6] indicates: the classes that are disjoint with the original input class should not be included.

This can easily be handled using the notion of *negative application condition* [10], that is an extension of the basic attributed graph transformation model. Basically, a negative application condition is a graph associated to a rule that, if isomorphic to the sub-graph matched by the premiss, prevent the rule from being applied. In other terms, a transformation rule can be seen as a tuple $R = \langle NAC_R, LHS_R, RHS_R, Map_R \rangle$ meaning that, if there exists a sub-graph SG of the source graph such that SG is isomorphic to LHR_R and SG *is not isomorphic to* NAC_R, then SG is replaced by RHS_R in the source graph, taking into account the mappings in Map_R.

In the case of the last technique, adding another boolean attribute **input** for identifying the input class[8], the *Negative Application Condition NAC* of the rule would be:

NAC

$$\begin{tikzcd}
\texttt{N}_2{:}\mathit{Class}\,(\mathbf{inc}, \mathbf{input}) \\
\texttt{N}_1{:}\mathit{Class} \arrow[u, "disjointWith"']
\end{tikzcd}$$

The *NAC* in that case acts as the expression of an exception to the rule.

4.3 For Knowledge Selection

[4] describes a module extraction tool integrated in a larger process, called knowledge selection, that aims at retrieving the relevant components from online ontologies, to be used in webpage annotation. The principle here is, like the Galen segmentation service, to include all the elements that participate to the definition of the included entities. However, compared to the previously mentioned technique, the one of [4] has two main particularities: First it makes use of inferences during the modularization process, meaning that the relations that are considered can be inferred relations as well as declared ones. Second, in order to reduce the size of the module (in particular, to facilitate its visualization), not all the super-classes of the included classes are included. The technique actually *takes shortcuts* in the class hierarchy by including only the named classes that are the most specific common super-classes of included classes.

Concerning the use of inferences during the modularization process, instead of taking as a source graph the representation of the elements declared in the ontology, we use the translation of the inferred elements. In particular, in that case, a *subClassOf* edge in the source graph no longer means that one class is directly declared to be a sub-class of the other, but also that it can be inferred, for example because of the transitivity of the sub-class relation. It is worth mentioning that an incomplete reasoning mechanism could be implemented as part of the graph transformation model, where *inference rules* on the ontology language would be translated as graph transformation rules. For example, the inferences linked to the fact that the *subClassOf* relation is transitive can be

[8] This attribute can also be specified for other techniques, even if not used.

represented by a rule generating new *subClassOf* relations in the graph, on the basis of existing ones. However, this would represent only a partial solution since this implementation of ontological reasoning would be very incomplete and inefficient compared to existing, tableau based reasoners.

Using the inferred graph, the first rule of the technique is the same as the one of the Galen segmentation service, except that it excludes named classes, which are treated separately.

Premiss	**NAC**	**Transformation**
$\begin{tikzcd} \mathtt{N}_2{:}* \\ \mathtt{N}_1{:}*\ (\mathbf{inc}) \arrow[u, "*"'] \end{tikzcd}$	$\begin{tikzcd} \mathtt{N}_2{:}\mathit{Class}\ (\mathbf{name}!=\text{""}) \\ \mathtt{N}_1{:}*\ (\mathbf{inc}) \arrow[u, "*"'] \end{tikzcd}$	$\begin{tikzcd} \mathtt{N}_2{:}*\ (\mathbf{inc}) \\ \mathtt{N}_1{:}*\ (\mathbf{inc}) \arrow[u, "*\ (\mathbf{inc})"'] \end{tikzcd}$

As explained above, concerning named classes, only the most specific common super-classes of included classes are included in this technique. The most specific super-class C of two classes A and B is a super-class of both A and B such that there is not any other super-class of A and B that is a sub-class of C. This can easily be represented by a combination of the (positive) premiss of the rule and of a negative application condition:

Premiss	**NAC**	**Transformation**
$\begin{tikzcd} & \mathtt{N}_3{:}\mathit{Class} \\ \mathtt{N}_1{:}\mathit{Class}\ (\mathbf{inc}) \arrow[ur, "\sqsubseteq"'] & \\ & \mathtt{N}_2{:}\mathit{Class}\ (\mathbf{inc}) \arrow[uu, "\sqsubseteq"'] \end{tikzcd}$	$\begin{tikzcd} & \mathtt{N}_3{:}\mathit{Class} & \\ & \mathtt{N}_4{:}\mathit{Class} \arrow[u, "\sqsubseteq"'] & \\ \mathtt{N}_1{:}\mathit{Class}\ (\mathbf{inc}) \arrow[ur, "\sqsubseteq"'] & & \mathtt{N}_2{:}\mathit{Class}\ (\mathbf{inc}) \arrow[ul, "\sqsubseteq"] \end{tikzcd}$	$\begin{tikzcd} & \mathtt{N}_3{:}\mathit{Class}\ (\mathbf{inc}) \\ \mathtt{N}_1{:}\mathit{Class}\ (\mathbf{inc}) \arrow[ur, "\sqsubseteq\ (\mathbf{inc})"'] & \\ & \mathtt{N}_2{:}\mathit{Class}\ (\mathbf{inc}) \arrow[uu, "\sqsubseteq\ (\mathbf{inc})"'] \end{tikzcd}$

Note that these two rules make use of the fact that the implicit, indirect sub-class relations are made explicit. Comparing the rules, and considering that the ones corresponding to the technique in [4] are more "selective" than the one of Galen, it seems obvious that it will result in smaller modules.

4.4 Prompt Traversal View Extraction

Prompt is a tool including several ontology manipulation and comparison features that are integrated as a plugin of the Protégé ontology editor. In particular, it includes a feature called *traversal view extraction*, which can be seen as an ontology module extraction technique [2]. With this tool, modules extracted from external ontologies are integrated into the ontology currently being edited.

Starting from one class of the considered ontology, Prompt *traverses* the relations of this class recursively to include related entities. In that sense, the principle is similar to the one applied in [3] and [4]. However, there are two main reasons that distinguish Prompt from these techniques. First, while [3, 4] intend to provide an automatic method, Prompt is designed as an interactive tool, allowing the user to incrementally build ontology modules by extending the currently extracted one. Second, Prompt allows the user to select the relations to be traversed and to associate to each of them a level of recursion, at which the algorithm should stop "traversing" the relation.

Whilst this first particularity of Prompt, its interactivity, affects only the overall system; the second one introduces a certain level of sophistication in the transformation rules required to implement it. In order to identify selected properties to be traversed, an attribute **selected** is associated to the node they correspond to. Another attribute is used to represent the maximum recursion level for each of the selected properties. Two rules are considered: one for the first step of the traversal, and one for the following steps.

Premiss	**Transformation**
$\begin{tikzcd} \mathtt{N_2}{:}* \\ \mathtt{N_1}{:}*\ (\mathbf{inc}) \arrow[u, "P"'] \end{tikzcd}$ $\mathtt{P}{:}\mathit{Propety}\,(\mathbf{selected}, \mathbf{maxrecur}{=}x)$	$\begin{tikzcd} \mathtt{N_2}{:}*\ (\mathbf{inc}) \\ \mathtt{N_1}{:}*\ (\mathbf{inc}) \arrow[u, "P\,(\mathbf{inc}, \mathbf{recur}{=}x{-}1)"'] \end{tikzcd}$ $\mathtt{P}{:}\mathit{Propety}\,(\mathbf{selected}, \mathbf{maxrecur}{=}x)$

Premiss	**Transformation**
$\begin{tikzcd} \mathtt{N_2}{:}* \\ \mathtt{N_1}{:}*\ (\mathbf{inc}) \arrow[u, "P"'] \\ \mathtt{N_3}{:}*\ (\mathbf{inc}) \arrow[u, "P\,(\mathbf{inc}, \mathbf{recur}{=}\ x\,(x!{=}0))"'] \end{tikzcd}$	$\begin{tikzcd} \mathtt{N_2}{:}*\ (\mathbf{inc}) \\ \mathtt{N_1}{:}*\ (\mathbf{inc}) \arrow[u, "P\,(\mathbf{inc}, \mathbf{recur}{=}x{-}1)"'] \\ \mathtt{N_3}{:}*\ (\mathbf{inc}) \arrow[u, "P\,(\mathbf{inc}, \mathbf{recur}{=}\ x)"'] \end{tikzcd}$

As we can see, this corresponds to a specialization of the Galen segmentation service rule, introducing a limitation. Indeed, the Prompt rule applied with all the properties selected and a level of recursion infinite would give the same result as Galen. Otherwise, the result would be smaller.

4.5 Locality Based Module Extraction

[5] defines a module as a *minimal, conservative extension* [11] of the original ontology with respect to the considered sub-vocabulary. Being a conservative extension means that the meaning of the terms of the input sub-vocabulary is completely captured by the module, as it is in the source ontology. In [5], Cuenca-Grau et al. also show that computing a minimal module considering the

definition they provide is undecidable, and describe two different approximations based on the notion of locality. The first method makes use of a reasoner to check the semantic locality of the axioms to be included. This procedure is decidable but, due to the use of a reasoning mechanism, it is also complex. The second one syntactically tests the locality of axioms and can be realized in polynomial time.

This technique clearly shows the limit of our framework based on graph transformation. Indeed, since the first method relies on the use of a reasoner to dynamically realise complex inferences for testing semantic locality, it could not be implemented as graph transformation rules. The second method is based on the syntactic inspection of the axioms of the ontology, looking in particular at the structure of the entities linked by these axioms. Therefore, it should be possible to design a set of transformation rules implementing the test of syntactic locality, and so, the extraction of a module based on this notion. However, the number of rules that are required is too numerous to make this approach really practical.

5 Towards a Parametric Modularization Tool

In the previous section, a number of module extraction techniques were reformulated using a common mechanism, graph transformation, parametrized by a specific set of transformation rules. This has shown that the approach is feasible for most of the existing techniques.

The ultimate goal of this work is to build a tool in which the user could select the appropriate technique for his application, and even reuse, combine or create rules from different approaches to build a customised modularization technique. Figure 2 gives an overview of the architecture of such a tool. The components to implement would be the ontology-to-graph and graph-to-ontology converters, in accordance with the model described in Section 3. As already mentioned, we can rely on existing tools and libraries of graph transformation engines. The ontology-to-graph and graph-to-ontology converters will be able to take an ontology file as input or interface with a triple store, such as Jena[9].

The techniques described in the previous section provide an initial pool of rules that can be reused and combined. It could, for example, be possible to add the recursion level of Prompt [2] to the techniques described in [4] or [6]. However, a more complete study of the interactions between rules is required, as some may be incompatible, or may result in the whole ontology being extracted as a module. In the same line of idea, important efforts are required to provide support for the editing and creation of rules specifically for module extraction. Graphical editors and mechanisms for the persistence of module extraction rules need to be developed.

Also, a natural progression for such a tool would be to allow the user to interactively build modules. It is unlikely that any one module extraction technique would extract a module that meets the requirements of the user and their

[9] `http://jena.sourceforge.net/`

particular application. Thus, in the same way as Prompt allows, once an initial module is created, to refine it by choosing a new starting point. It would be interesting to build a module interactively, and iteratively, letting the user choose a different sub-vocabulary and a different set of transformation rules at each step. In this way, by controlling how the elements of the original ontology are to be included in the extracted modules, the user could build modules that match the requirements of their application.

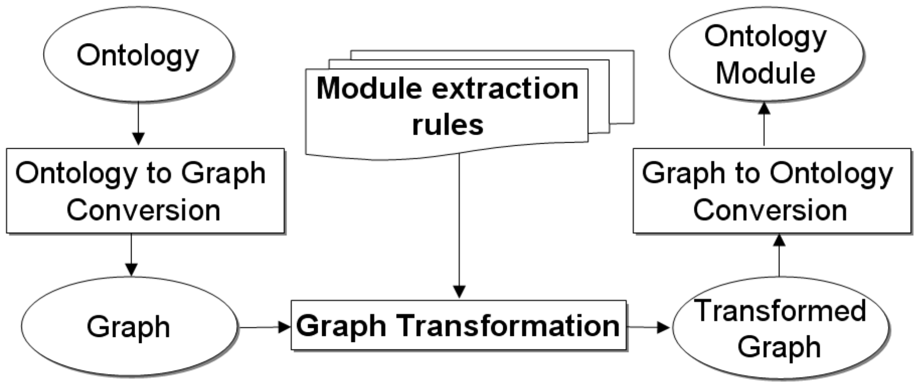

Fig. 2. Overview of the architecture of a parametric modularization tool based on graph transformation.

6 Conclusion and Future Work

In this paper, we have shown that, even if they are based on different assumptions and implementations, most of the ontology module extraction techniques rely on the same principles and can be reformulated in a common framework based on graph transformation. By reformulating existing techniques as graph transformation rules, the feasibility of such a framework has been shown. The interests of this work are multiple. First, having modularization techniques gathered together in the same format facilitates their comparison and evaluation. It is simpler for the user to understand them, match them to his requirements, and apply them. Second, this leads to the development of a parametric modularization tool, in which the parameters are the rules to apply for module extraction. The reformulated techniques provide an initial pool of rules to chose from and to combine to build the adequate technique in a given application scenario. In addition, this tool would give the possibility to easily (visually, graphically) build entirely new modularization techniques by editing, modifying and creating new module extraction rules, enabling application developers to obtain specialised modularization techniques, without having to re-implement a new tool or modify an existing one.

The obvious next step of this work is the implementation of such a tool, requiring the consideration of a number of different aspects, as described in Section 5. Moreover, in order to facilitate the exploitation of this tool, and the manipulation of modularization techniques as graph transformation rules, this parametric modularization tool should be integrated within an ontology development environment, such as Protégé.

Whilst reformulating the existing techniques into the graph transformation framework presented in this paper, it became clear that some elements and features are not easily represented in graph transformation rules. In particular, when trying to reformulate [5] it appeared that it was practically unfeasible to represent the test at the basis of this technique, the locality test, within our model. In order to overcome this limitation, it would be interesting to introduce *externally evaluated predicates* for nodes and edges, in addition to attributes. Such a predicate could be use for example to call an external reasoner to check for the locality of the considered axioms, and so implement the considered technique in an hybrid way.

Another interesting direction for future work would be to consider the reformulation of the partioning techniques, such as [12], [13], within the framework presented in this paper. Partitioning techniques are more complicated in the sense that they produce several modules, and are based on more complex, less localized computations.

A full comparison between the set of rules needed for each reformulated technique will be completed. This will, hopefully, provide a solid theoretical comparison that would allow the user to better select which approach best fits their needs.

Also, there is the need to consider scalability issues. With the framework presented in this paper it is assumed that the whole ontology will be transformed into a graph, which does not make sense in the cases where modularization is used as a way to improve the performance of ontology tools, by considering only a part of the original ontology. One possible solution to this scalability problem would be to build the graph on the fly, as transformation rules are applied, rather than building the whole graph in the beginning and then doing the extraction.

References

1. Parnas, D.: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM **15** (1972)
2. Noy, N., Musen, M.: Specifying Ontology Views by Traversal. In: Proc. of the International Semantic Web Conference (ISWC). (2004)
3. Seidenberg, J., Rector, A.: Web Ontology Segmentation: Analysis, Classification and Use. In: Proc. of the World Wide Web Conference (WWW). (2006)
4. d'Aquin, M., Sabou, M., Motta, E.: Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies. (2006)

5. Cuenca-Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: Extracting modules from ontologies. In: Proc. of the 16th International World Wide Web Conference (WWW 2007), pages 717-727, Banff, AB, Canada, May 812 2007. ACM Press. (2007)
6. Doran, P., Tamma, V., Iannone, L.: Ontology module extraction for ontology reuse: An ontology engineering perspective. In: Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management. (2007)
7. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M.: Ontology Modularization for Knowledge Selection: Experiments and Evaluations. In: 18th International Conference on Database and Expert Systems Applications - DEXA'07. (2007)
8. Ehrig, H., Prange, U., Taentzer, G.: Fundamental Theory for Typed Attributed Graph Transformation. In: 2nd International Conference on Graph Transformation. (2004)
9. Taentzer, G.: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: Applications of Graph Transformations with Industrial Relevance. Springer (2004)
10. Habel, A., Heckel, R., Taentzer, G.: Graph Grammars with Negative Application Conditions. Fundamenta Informaticae **26** (1996) 287–313
11. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In Veloso, M.M., ed.: IJCAI. (2007) 453–458
12. Cuenca-Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Automatic partitioning of owl ontologies using e-connections. In: Proceedings of the 2005 International Workshop on Description Logics (DL-2005). (2005)
13. Stuckenschmidt, H., Klein, M.: Structure-based partitioning of large concept hierarchies. In: Proceedings of the 3rd International Semantic Web Conference. (2004)