Structuring abstraction to achieve ontology modularisation

Zubeida C. Khan
Council for Scientific and Industrial Research, South Africa

C. Maria Keet
Department of Computer Science, University of Cape Town, South Africa

ABSTRACT

Large and complex ontologies lead to usage difficulties, thereby hampering the ontology developers' tasks. Ontology modules have been proposed as a possible solution, which is supported by some algorithms and tools. However, the majority of types of modules, including those based on abstraction, still rely on manual methods for modularisation. Toward filling this gap in modularisation techniques, we systematised abstractions and selected five types of abstractions relevant for modularisation for which we created novel algorithms, implemented them, and wrapped it in a GUI, called NOMSA, to facilitate their use by ontology developers. The algorithms were evaluated quantitatively by assessing the quality of the generated modules. The quality of a module is measured by comparing it to the benchmark metrics from an existing framework for ontology modularisation. The results show that module's quality ranges between average to good, whilst also eliminating manual intervention.

Keywords: Ontology, Semantic Web, Ontology Modularisation, Ontology Module, Ontology Modularity, Abstraction

INTRODUCTION

An ontology is a logic-based model that is used to represent a subject domain in a machine-processable language for Semantic Web applications. There are various formal definitions presented in the literature for an ontology. A comprehensive definition for an ontology is as follows: (Guarino, 1998) :

> *An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models.*

Ontologies are commonly used for some purpose such as enhanced searches across many databases or decision-making support for problems. It consists of entities, organised in a hierarchy, with relations among them and axioms to define such relations. The entities are often constrained and described using the other entities to provide better meaning.

Ontologies that describe large, well-defined domains are consequently large and complex in nature; e.g., the NCBI Taxonomy with its 1,015,206 classes (Federhen, 2012), and likewise SNOMED CT (Donnelly,

2006) and the FMA (Rosse & Mejino, 2003) are well-known to be challenging due to their size. This leads to difficulties for tools and humans alike: tools cannot process them due to computational limitations while humans face cognitive overload for understanding them. Over the last few years, there has been a growth in using modularity to assist with large ontologies (Amato et al., 2015; d'Aquin et al., 2006; Del Vescovo, 2011; Grau et al., 2008; Khan & Keet, 2016b). A module has been defined as follows (Khan & Keet, 2015):

> *A module M is a subset of a source ontology O, M ⊂ O, either by abstraction, removal or decomposition, or module M is an ontology existing in a set of modules such that, when combined, make up a larger ontology. Module M is created for some use-case U, and is of a particular type T. T is classified by a set of annotation features P, and is created by using a specific modularization technique MT, and has a set of evaluation metrics EM which is used to assess the quality of module M.*

The general idea of modularisation refers to dividing and separating the components of a large system such that it can be recombined into a whole. The purpose of modularity is used to simplify and downsize an ontology for the task at hand, i.e., to modularise a large ontology into smaller ontologies, and it is required in ontology development and use when one needs to hide or delete knowledge that is not required for the use-case (Studer, 2010). Modularisation has been applied to various ontologies to improve usability and assist with complexity; e.g., the myExperiment ontology (Newman et al., 2009), the Semantic Sensor Net ontology (Janowicz & Compton, 2010), several BioPortal ontologies (Del Vescovo et al., 2011), and the FMA ontology (Mikroyannidi et al., 2009).

There are several modularisation methods and tools (Amato et al., 2015; Chen et al., 2019; d'Aquin et al., 2006; Hamdi et al., 2010; Kalyanpur et al., 2006; LeClair et al., 2019). For ontologies that are created in a modular way, however, many of them are created by manual methods (Khan & Keet, 2015). There is tool support for graph partitioning (Kalyanpur et al., 2006), query-based (Natalya Fridman Noy & Musen, 2000) and locality-based (Grau et al., 2008) techniques. We zoom in on the type of modules obtained from abstractions, taken from an existing categorisation of ontology modules (Khan & Keet, 2015). Abstraction is the principle of simplifying complex models by removing some unnecessary details based on some criteria, such as reducing a class hierarchy's depth or removing axioms to fit the ontology in a language of lower expressiveness. The purpose of abstraction, like modularity, is to have a simplified version of an ontology for a specific task or application.

Seeing that automatic techniques for generating these types of modules are lacking (discussed below), we solve this problem by investigating and structuring proposed types of abstractions, which led to the creation of a basic theory from the related works. Based on the theory, several abstractions were selected to fill in the gaps for the lacking tool support and we propose new algorithms for generating abstraction and expressiveness modules. The algorithms have been implemented and are thus the first fully automated abstractions for ontologies. They have been evaluated on adequacy with 128 ontologies. The generated modules have an average to good quality according to typical modularity metrics. The core implementation is also wrapped in a GUI called NOMSA (Novel Ontology Modularisation SoftwAre) for

ease of used by the ontology engineer. The objective of the work is two-fold: 1) To create a basic theory of abstraction for ontology modules, and 2) To provide automated tools for generating such modules.

The remainder of the chapter is structured as follows. In the background section, we discuss related works on modularity. This is followed by a section on structuring abstraction towards a theory, which is followed by our new approaches for modularisation. We then present the implementation and experimental evaluation of the algorithms section and a discussion section. Lastly, we provide a conclusion section.

BACKGROUND

There are three broad techniques for creating modules from a larger ontology, including techniques such as graph partitioning, logic-based approaches for locality, and queries. Yet, there is also a heavy reliance on manual methods for module creation: manual methods were used for module creation for 9 out of the 14 module types identified in the framework for ontology modularisation (Khan & Keet, 2015). One well-known approach to create modules, which is also heavily reliant on manual methods, is abstraction, which looks at simplifying an ontology in some way. Several different types of abstraction have been identified and formalised for logical theories and ontologies, notably (Ghidini & Giunchiglia, 2004; Keet, 2005, 2007; Mani, 1998; Pandurang Nayak & Y. Levy, 1995). For instance, one can generalise by exploiting subsumption along the class hierarchy (Ghidini & Giunchiglia, 2004; Keet, 2007; Pandurang Nayak & Y. Levy, 1995), or along a partonomy, among other relations, and 'relevance'-based deletions, however 'relevance' may be specified (Keet, 2007). These types are promising for achieving abstraction, executing it and evaluating whether the resultant module is indeed a good one, but they either require additional user input or have not been implemented.

The notion of abstracting models also has been looked into for managing large conceptual models, such as for Object-Role Modeling (Campbell et al., 1996; Keet, 2005) and Entity-Relationship diagrams (Jaeschke et al., 1994). Their core approach is to use 'relevance' measures by weighting language features. This cannot be reused directly for ontologies, as there are differences in language features and also those rules have not been implemented. To a certain extent related to weighting the language features, is the interplay between an ontology in, say OWL 2 DL and simplifying it into one represented in an OWL Profile (sub-language), or any combination of more/less expressive languages, giving rise to (OWL) 'profile modules' (e.g., (Krötzsch, 2012)) and Protégé v5 has a feature for generating modules in some OWL Profiles[1]. The recently standardised DOL has taken this notion for expressiveness-based networks of modules as its core feature (Mossakowski et al., 2015).

Abstractions for other types of models and languages have been proposed as well (see (de Lara et al., 2013) for an overview), but they have even less features in common with ontologies. De Lara et al.'s (de Lara et al., 2013) four high-level types of abstraction—aggregation, merge, deletion, and view generation—have been proposed also for ontologies and conceptual models, but not the metamodel approach with "structural concepts", for there is less heterogeneity in ontology languages.

[1] https://protegewiki.stanford.edu/wiki/P4MoveAxioms#Refactor

Thus what abstraction entails for ontologies varies in the literature, and little has been implemented so as to examine the outcome of such abstraction operations beyond the manual examples given.

STRUCTURING ABSTRACTIONS TOWARD A THEORY FOR ABSTRACTION

Informed by the related works, we structure, clarify, and refine those notions of abstraction. Description Logics (DLs) is a formalization that may be used for representing ontologies which is used in the remainder of the Chapter. We assume that the reader is familiar with DLs.

We do not claim that this is an exhaustive list ontologically, but is exhaustive with respect to current proposals. Also, we cast the net for 'abstraction' wide, as in (Fridman et al., 2015): the outcome of an abstraction operation is a simpler artefact (ontology/ logical theory/ conceptual model), or view thereof, that retains certain elements or features of the original one but not all. Cognitively, in the general case, the 'simpler' is expected to be easier to understand because of the lower complexity and should be in some way a smaller artefact than the original. Ontologically, one intentionally removes or ignores certain finer-grained details of the representation of reality, or one's conceptualization thereof, in the abstraction process, i.e., it does not involve denying that certain details exist, but concerns the choice not to deal with that for some reason. We identified the following abstractions with respect to ontologies and related artefacts at the same stratum[2].

1. Abstraction along a hierarchy:
    a. Along a subsumption/class taxonomy: an entity is abstracted into its parent entity (e.g., (Ghidini & Giunchiglia, 2004; Keet, 2007; Pandurang Nayak & Y. Levy, 1995));
    b. Along a partonomy (mereology): an entity is abstracted away into the whole that it is part of (Keet, 2007);
    c. Along a meronymic relation that is not parthood: an entity is abstracted away into the whole that it is part of, such as participation, membership, and constitution (Keet, 2007; Mani, 1998);
    d. Along an arbitrary transitive relation in a domain ontology;
2. Abstraction along axioms with relations:
    a. Along relation chains or composition: this 'shortcuts' a chain/joins; e.g., for $R \circ S \sqsubseteq T$, to keep T and ignore or delete R and S (where $R \neq S \neq T$);
    b. Along a class definition, also called folding (Keet, 2007; Mani, 1998), such as removing a class's attributes and relations to other classes;
3. Abstraction based on 'relevance', for which there are various ways in which certain entities are deemed more important than others:

[2] That is, we exclude other notions of abstractions in different contexts, e.g., those when needing to deal with going from object in reality to its computerised representation or from individual to its class or set.

a. Manual identification of important entities for the subject domain that must remain in the ontology; e.g., in an ontology of pets, the popular ones—cats, dogs—are more important than, say, geckos and snakes (Schulz & Boeker, 2013);

b. Relative well-connected important entities in the ontology, i.e., those entities participating in a lot of axioms, receive higher weighting cf. the less used ones and orphan entities that are then abstracted away (Natalya F Noy & Musen, 2009);

c. Syntax-based rules where one language feature is deemed more important for the represented knowledge than another (Campbell et al., 1996; Jaeschke et al., 1994; Keet, 2005); e.g., to provide more weight to existentially quantified relations over universally quantified ones, and therewith the classes involved in it obtain more points as being more important and thus survive the abstraction process;

d. Hierarchy depth where entities higher up in the hierarchy are deemed more relevant and, e.g., removing everything below a specified depth (de Lara et al., 2013);

4. Abstraction motivated by language expressiveness, i.e., driven by some ontology language's features:

   a. Language feature deletion: simply delete any axiom that uses the offending language features to obtain a module in a less expressive language (Krötzsch, 2012);

   b. Approximation: use a set of rewriting rules to approximate the subject domain semantics of the axiom(s) that is(are) deleted, which typically increases the number of axioms (Botoeva et al., 2010);

   c. Language vocabulary elements: this involves removing certain elements that may be unnecessary for a certain use-case; e.g., data properties.

For this work, we focus on designing algorithms to match those that are lacking for certain module types from an existing ontology modularisation framework and experiment (Khan & Keet, 2015). From the experiment, it was found that for several module types, manual methods were used. We wish to automate the methods for abstraction and expressiveness type modules by selecting the following abstractions to design algorithms for: 2.Generalisation along axioms (b), 3. 'Relevance'-based abstraction (b, d), and 4.Language expressiveness (a, c).

MODULARISATION METHODS INVOLVING ABSTRACTIONS

Within the scope of methods for modularisation, we focus on automating the methods used for generating abstraction modules, based on the types of modules identified in (Khan & Keet, 2015).There are four types of abstraction modules and one type of expressiveness module. To fill this gap, we create algorithms for five types of abstraction, which are listed in Table 1.

We now define and introduce the algorithms for these types. We formally define each of the types of modules together with its corresponding algorithm. The class of an ontology is mentioned in several definitions, and we are referring to the set of axioms describing a class; the same holds for the object property and data property of an ontology.

Table 1: The algorithms that are created for each of the abstractions

Algorithm	Abstraction
AxAbs	2b. Generalisation along axioms with relations: Along a class definition
VocAbs	4c. Language expressiveness: Along vocabulary elements
HLAbs	3d. Relevance -based abstraction: Hierarchy depth
WeiAbs	3b. Relevance -based abstraction: Relative well-connected important entities
FeatExp	4a. Language expressiveness: Language feature deletion

Axiom abstraction (Algorithm 1 (AxAbs)) generates a module without relations between entities; therefore, the technique decreases the horizontal structure of the ontology and makes it a bare taxonomy. Axiom abstraction is formally defined as follows:

Definition 1 (Axiom Abstraction) *Let $\mathcal{O}, \mathcal{O}'$ be two ontologies. $\mathcal{S} = \{\alpha_1, \ldots, \alpha_k\}$ a set of axioms involving at least two classes or a class and a data type, and either at least one object property or data property from $\mathcal{O}$ (i.e., GCIs). We say that $\mathcal{O}'$ is an axiom abstraction module of $\mathcal{O}$, if $\mathcal{O}' \cup \mathcal{S} = \mathcal{O}$ such that there exists no element of $\mathcal{S}$ in $\mathcal{O}'$ (i.e., $\mathcal{S} \cap \mathcal{O}' = \emptyset$, hence $' \subset \mathcal{O}$.*

For instance, if `Professor` $\sqsubseteq$ $\exists$`teaches.Course` $\in \mathcal{S}$ (as it is not a simple subsumption between named classes nor is it a declaration of the Professor class; line 4 of AxAbs), then the axiom will be removed (lines 8-10) resulting in module $\mathcal{O}'$ that will contain just the classes `Professor` and `Course`.

Algorithm 1: Axiom abstraction to compute module *M* (AxAbs).

```
1:Input Ontology O
2:Output Module M
3:for all axiom ∈ O do
4:    if axiom.type==subclass_axiom or axiom.type==declaration_axiom then
5:          cExpression←axiom.getNestedClassExpressions()
6:          cExpressionSet←cExpressionSet+cExpression
            // cExpressionSet is a data structure where we store all the
            class expressions
7:                for all cExpression ∈ cExpressionSet do
8:                      if cExpression.type ≠class then
9:                            remove axiom
10:                     end if
11:               end for
12:   else
13:         remove axiom
14:   end if
15:end for
16:M←O
```

Applying vocabulary abstraction to an ontology generates a module where a certain vocabulary element is removed from the ontology.

Definition 2 (Vocabulary abstraction) *Let $\mathcal{O}, \mathcal{O}'$ be two ontologies, $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ the set of classes in $\mathcal{O}$, $\mathcal{OP} = \{\mathcal{OP}_1, \ldots, \mathcal{OP}_k\}$ the set of object properties in $\mathcal{O}$, and $\mathcal{DP} = \{\mathcal{DP}_1, \ldots, \mathcal{DP}_k\}$ the set of data properties in $\mathcal{O}$. We say that $\mathcal{O}'$ is a vocabulary abstraction module of $\mathcal{O}$, if $\mathcal{O}' \cup \mathcal{C} = \mathcal{O}$ or $\mathcal{O}' \cup \mathcal{OP} = \mathcal{O}$ or $\mathcal{O}' \cup \mathcal{DP} = \mathcal{O}$ such that there exists no element of $\mathcal{C}$, $\mathcal{OP}$, or $\mathcal{DP}$ in $\mathcal{O}'$ (i.e., $\mathcal{C} \cap \mathcal{O} = \emptyset$, $\mathcal{OP} \cap \mathcal{O} = \emptyset$, $\mathcal{DP} \cap \mathcal{O} = \emptyset$), hence $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 2: Vocabulary abstraction to compute module *M* (VocAbs).

```
1:Input Ontology O, element e, element type t
// e is a named class, object property, or data property and t is the type of e in O
2:Output Module M
3:    if t==class then
4:          remove e
5:    else if t==object property then
6:          remove e
7:    else if t==data property then
8:          remove e
9:    else if t==individual then
10:         remove e
11:   end if
12:M←O
```

For instance, consider a version of the Infectious Disease Ontology (IDO) (Cowell & Smith, 2009) that is linked to the BFO foundational ontology and the developers want to change that to DOLCE for interoperability in a heterogeneous system. Since the domain ontology does not contain any object properties, one could be interested in removing the object properties from the DOLCE-aligned version of the IDO ontology using the vocabulary abstraction algorithm.

High-level abstraction generates a module where entities at a higher level are regarded more important than others (Algorithm 3 (HLAbs)), and is defined as follows, specifying the notion of depth in a taxonomy first:

Definition 3 (Depth) *Let $\mathcal{O}$ be an ontology. A depth in the hierarchy of $\mathcal{O}$ represents the subclass distance between the asserted hierarchy's top-level entity and a given entity; e.g., depth 1 refers only to the top-level classes, depth 2 refers to the top 2 layers of classes (the parent classes and its direct subclasses) and so on.*

Definition 4 (High-level abstraction) *Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, n be a depth where n is an integer ≥ 1. We say that $\mathcal{O}'$ is a High-level abstraction module of $\mathcal{O}$, if the entities with a depth > n are removed, hence, $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 3: High-level abstraction to compute module *M* (HLAbs).

```
1:Input Ontology O, levelNumber
2:Output Module M
3:oldSet←ontology.getAxioms()
```

```
4:for all class ∈ O do
5:    if class.superclasses() is empty then
6:          topLevelClassSet←topLevelClassSet+class
7:    end if
8:end for
9:counter←0
10:while counter ≠ levelNumber do
11:   for all topclass ∈topLevelClassSet do
12:         newset←newset+topclass.getAxioms()
13:               if topclass.subclasses()≠ empty then
14:                     temp←topclass.subclasses()
15:               end if
                  //Repeat lines 3 - 13 for object properties and data
                  properties.
16:   topLevelClassSet.clear()
17:   topLevelClassSet←temp
18:   end for
19:end while
20:for all axiom ∈oldAxioms do
21:   if newAxioms does not contain axiom then
22:         ontology.remove(axiom)
23:   end if
24:end for
25:M←O
```

For instance, the GFO-abstract-top module of the GFO ontology in the ROMULUS repository (Khan & Keet, 2016a) is based on the Abstract Top Level layer of GFO which contains mainly two meta-categories: set and item. This module can be generated automatically with HLAbs by setting the depth to 2 (see Figure 1).

Weighted abstraction deals with removing entities from an ontology that are deemed less important than others by assigning weight to the classes, properties, and individuals in an ontology. Our approach for determining this is based on examining entities that other entities are highly dependent on. For instance, in the pizza ontology, the class `TomatoTopping` is the most widely used, being referenced 61 times by other entities. Weighted abstraction is formally defined in Definition 7, availing of the notions of relative and absolute thresholds, which are introduced first.

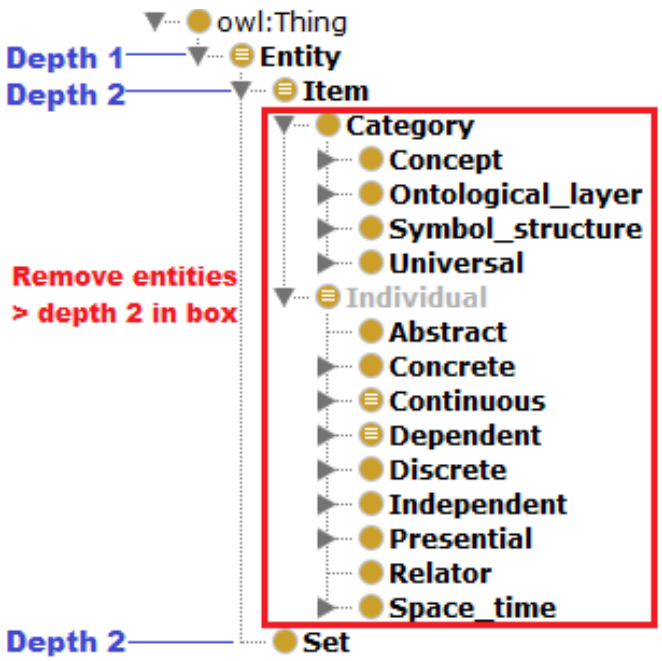

Figure 1: Generating a high-level abstraction module with depth = 2 from the GFO ontology.

Definition 5 (Relative threshold) *Let $\mathcal{O}$ be an ontology, $\mathcal{E} = \{\mathcal{E}_1,...,\mathcal{E}_k\}$ be the set of classes, object properties, data properties, and individuals in $\mathcal{O}$. A relative threshold θ is a percentage value to decide which elements of $\mathcal{E}$ are to be removed from $\mathcal{O}$. Each element of $\mathcal{E}$ is weighted according to the number of axioms it participates in and ordered according to a position p. If $p(\mathcal{E}_i) < \theta$, $\mathcal{E}_i$ is removed from $\mathcal{O}$.*

Definition 6 (Absolute threshold) *Let $\mathcal{O}$ be an ontology, $\mathcal{E} = \{\mathcal{E}_1,...,\mathcal{E}_k\}$ be the set of classes, object properties, data properties, and individuals in $\mathcal{O}$. An absolute threshold θ is a numerical value to decide which elements of $\mathcal{E}$ are to be removed from $\mathcal{O}$. Each element of $\mathcal{E}$ is weighted according to the number of axioms it participates in and ordered according to a position p, If $p(\mathcal{E}_i) < \theta$, $\mathcal{E}_i$ is removed from $\mathcal{O}$.*

Definition 7 (Weighted abstraction) *Let $\mathcal{O}$, $\mathcal{O}$ 'be two ontologies, $\mathcal{E} = \{\mathcal{E}_1,...,\mathcal{E}_k\}$ be the set classes, object properties, data properties, and individuals in $\mathcal{O}$. We say that $\mathcal{O}'$ is a weighted abstraction module of $\mathcal{O}$, if elements of $\mathcal{E}$ are removed from $\mathcal{O}$ according to some absolute threshold or relative threshold, hence, $\mathcal{O}' \subset \mathcal{O}$.*

Algorithm 4 (WeiAbs) generates weighted abstraction modules. For instance, consider modularising the BioTop ontology (Beisswanger et al., 2008) using the weighted abstraction algorithm with $\theta = 4$. For the class `Phosphate`, it has two referencing axioms: a declaration axiom and the axiom `Phosphate` $\sqsubseteq$ `InorganicMolecularEntity`. Since the number of referencing axioms ($p(\mathcal{E})$ =2) is less than the absolute threshold value (4), the `Phosphate` class is removed from the ontology.

Algorithm 4: Weighted abstraction to compute module *M* (WeiAbs).

```
Input Ontology O, thresholdPercentage, Weight_array, Class_array
Output Module M
i←0
for all class ∈ O do
```

```
5:    Weight_array←Num_of_referencing_axioms
6:    Class_array(i)←class
7:    i←i+1
8:end for
9:Sort(Weight_array,Class_array) //Sort Weight_array from low to high, with
Class_array corresponding to Weight_array
10:limit←thresholdPercentage*|Class_array|
11:for i←0,limit do
12: remove Class_array(i) from O
13:end for //repeat lines 4-12 for ObjectProperty_array,
DataProperty_array,Individual_array
14:M←O
```

Feature expressiveness modules deal with removing some axioms of the ontology based on the language features, e.g., cardinality constraints, disjointness, object property features etc. By manipulating complex constructs of the ontology language features, the feature expressiveness algorithm results in a simplified model of the ontology that maybe more suitable for scalability of the ontology-driven information system. We have designed seven rules for this to demonstrate it (discussed below). The algorithm takes these seven rules and removes them from the least important to the most important. At each rule removal, a 'layer' of the ontology is produced where that ontology is represented in a language of lower expressivity than the previous layer. Once the algorithm is complete, seven modules (layers) are produced, each having a lower level of expressivity than the previous. Feature expressiveness is formally defined as follows:

Definition 8 (Feature expressiveness) *Let $\mathcal{O}$, $\mathcal{O}'$ be two ontologies, $\mathcal{R} = \{\mathcal{R}_1,...,\mathcal{R}_k\}$ a set of rules describing various OWL language features. We say that $\mathcal{O}'$ is a feature expressiveness module of $\mathcal{O}$, if we remove axioms that follow $\mathcal{R}$ from $\mathcal{O}$, hence $\mathcal{O}' \subset \mathcal{O}$.*

We decided to assign lower points to those OWL ontology features that serve to restrict and refine entities such as cardinality and property characteristics. We assign higher points to disjointness, equality and inequality, and complex classes since they can be used in conjunction to define new classes. Rule 7 and rule 6 are concerned with OWL features pertaining to class creation, which is why we weight them as the most important; classes are the building blocks in an ontology. Rule 5 is concerned with instance data in an ontology which follows in terms of weighting. Rules 4 to rule 1 are concerned with specialising or refining the existing concepts in the ontology. This weighting is subjective and motivated by the modelling perspective on language features. It is conceivable, and possible if desired, to assign different weights to them; e.g., such that they are motivated by, and aligned with, the various OWL profiles.

In the notation of the rules that follow for the axiom types, *C, D, E* are class descriptions that may be simple (named classes) or complex, *R, S* are object properties, *U, V* are data properties, and *a, b, c* individuals in the vocabulary of the ontology, *n* is a non-negative integer, and all declared knowledge adheres to the OWL 2 DL syntax.

- R1: **Qualified cardinality** in an ontology has an assigned weight of 1 point. *Rules: Remove axioms of the following axiom type, if present:* $C \sqsubseteq \leq n\ R.D$, $C \sqsubseteq \geq n\ R.D$, $C \sqsubseteq = n\ R.D$, $C \sqsubseteq \leq n\ U.D$, $C \sqsubseteq \geq n\ U.D$, $C \sqsubseteq = n\ U.D$.

- R2: **Domain and range** weigh 2 points. *Rule: Remove axioms of the following axiom type, if present:* $\exists R.\top > \sqsubseteq C$, $\top \sqsubseteq \forall R.C$.
- R3: **Object Property characteristics** weighs 3 points. *Rule: Remove axioms of the following axiom type, if present (in SROIQ's shorthand notation):* `Func`(*R*), `Func`(*R*−), `Sym`(*R*), `Asym`(*R*), `Trans`(*R*), `Ref`(*R*), `Irr`(*R*).
- R4: **Disjointness** weighs 4 points. *Rule: Remove axioms of the following axiom type, if present:* $C \sqcap D \sqsubseteq \bot$, $C \sqsubseteq \neg D$.
- R5: **Assertions** weigh 5 points. *Rule: Remove axioms of the following axiom patterns, if present: a: C, R(a,b), U(a,c).*
- R6: **Atomic equivalence and equality** weighs 6 points. *Rule: Remove axioms of the following axiom type, if present:* $C \equiv D$, $R \equiv S$, $U \equiv V$, $a = b$, $a \neq b$.
- R7: **Complex classes** are the most important of the seven rules, which we weigh with 7 points. Rule: *Remove axioms of the following axiom patterns, if present:* $C \sqsubseteq D \sqcap E$, $C \equiv D \sqcap E$, $C \sqsubseteq D \sqcup E$, $C \equiv D \sqcup E$.

Algorithm 5 (FeatExp) uses these rules to generate feature expressiveness modules. For instance, when modularising the BioTop ontology (Beisswanger et al., 2008), for rule 2, concerning the domain, the axiom $\exists$`processualQuality.`$\top \sqsubseteq$ `Quality` is removed from the module, whilst keeping the entities `Quality` and `processualQuality` in the module.**Algorithm 5:** Feature Expressiveness to compute module *M* (FeatExp).

```
1:Input Ontology O, ruleSet{r1,..r7}
2:Output Module moduleSet{M1,..M7}
3:i←1
4:for all axiom ∈ O do
5:    for all ri ∈ ruleSet do
6:          if axiom.type is ri then
7:                remove axiom
8:          end if
9:    end for
10:Mi←O
11:i←i+1
12:end for
```

Finally, note that the algorithms are linear for VocAbs and WeiAbs, and quadratic for AxAbs, HLAbs and FeatExp.

ILLUSTRATION OF ALGORITHMS

We now illustrate the algorithms introduced in the previous section with a sample ontology, where we focus on the WeiAbs and FeatExp algorithms. Consider the following axioms in a toy Burger ontology in Figure 2 (entity declaration axioms omitted).

$$\text{BeefPatty} \sqsubseteq \text{Patty} \quad (1)$$
$$\text{Beefburger} \equiv \text{HamBurger} \quad (2)$$
$$\text{Beefburger} \sqsubseteq \text{Burger} \quad (3)$$
$$\text{Cheapburger} \sqsubseteq \leq 1\ \text{hasFilling.Filling} \quad (4)$$
$$\text{Cheapburger} \sqsubseteq \text{Burger} \quad (5)$$
$$\text{Cheese} \sqsubseteq \text{Filling} \quad (6)$$
$$\text{Chef} \sqsubseteq \text{Person} \quad (7)$$
$$\text{Customer} \sqsubseteq \text{Person} \quad (8)$$
$$\text{HamBurger} \equiv \text{Beefburger} \quad (9)$$
$$\text{HamBurger} \sqsubseteq \text{Burger} \quad (10)$$
$$\text{HealthyBurger} \sqsubseteq \forall \text{hasFilling.(Lettuce} \sqcup \text{Tomato)} \quad (11)$$
$$\text{HealthyBurger} \sqsubseteq \text{Burger} \quad (12)$$
$$\text{Lettuce} \sqsubseteq \text{Filling} \quad (13)$$
$$\text{Medium} \sqsubseteq \text{PattyCook} \quad (14)$$
$$\text{PattyCook} \equiv \text{Medium} \sqcup \text{Rare} \sqcup \text{WellDone} \quad (15)$$
$$\text{Rare} \sqsubseteq \text{PattyCook} \quad (16)$$
$$\text{Sauce} \sqsubseteq \text{Filling} \quad (17)$$
$$\text{Tomato} \sqsubseteq \text{Filling} \quad (18)$$
$$\text{WellDone} \sqsubseteq \text{PattyCook} \quad (19)$$
$$\text{WhiteBun} \sqsubseteq \text{BurgerBun} \quad (20)$$
$$\text{WholeWheatBun} \sqsubseteq \text{BurgerBun} \quad (21)$$
$$\text{WholeWheatBun} \sqsubseteq \neg \text{WhiteBun} \quad (22)$$
$$\text{Func(hasBun)} \quad (23)$$
$$\exists\ \text{hasBun.}\top \sqsubseteq \text{Burger} \quad (24)$$
$$\top \sqsubseteq \forall \text{hasBun.BurgerBun} \quad (25)$$
$$\exists\ \text{hasPatty.}\top \sqsubseteq \text{Burger} \quad (26)$$
$$\top \sqsubseteq \forall \text{hasPatty.Patty} \quad (27)$$
$$\exists\ \text{hasPattyCook.}\top \sqsubseteq \text{Patty} \quad (28)$$
$$\top \sqsubseteq \forall \text{hasPattyCook.PattyCook} \quad (29)$$
$$\text{MarthasBurger} \neq \text{MyBurger} \quad (30)$$
$$\text{MarthasBurger} : \text{Burger} \quad (31)$$
$$\text{MyBurger} : \text{Beefburger} \quad (32)$$
$$\text{MyBurger} : \text{Burger} \quad (33)$$
$$\text{MyBurger} : \text{Beefburger} \quad (34)$$
$$\text{ChefRose} : \text{Chef} \quad (35)$$
$$\text{cookedBy(MyBurger, ChefRose)} \quad (36)$$

Figure 2: The burger ontology to which the algorithms are applied; see text for details.

To generate a weighted abstraction module, we apply WeiAbs. Let us assume we wish to create a module whereby we remove 25% of the entities. To achieve this, we set the threshold value to 25%. First, we apply lines 4-8 of WeiAbs, where we weigh each class in the ontology with its number of referencing axioms and we store both the number of referencing axioms and each class in two arrays with corresponding indices. For line 9 of the algorithm, we sort the weight array values from low to high and the class array such that it matches the weight array. In line 10, a limit variable is calculated as the product of the threshold percentage (.25) and the number of classes in the ontology (21) which is rounded off to a value of 5. In lines 11-13, the classes with the 5 lowest values are removed, as displayed in Table 2. The classes in bold font are the 25% of the classes that are deemed less-important than the rest and are to be removed due to having the least number of referencing axioms in the ontology.

Table 2: The classes of the burger ontology with the number of referencing axioms. Those in bold font are the classes to be removed for the resulting module

WhiteBun	**2**	Medium	3	Patty	4
Customer	**2**	Lettuce	3	BeefBurger	4
Cheese	**2**	HealthyBurger	3	BurgerBun	4
Sauce	**2**	BeefPatty	3	Hamburger	4
Chef	**2**	Tomato	3	Filling	5
WholeWheatBun	**2**	WellDone	3	PattyCook	6
Person	3	Rare	3	Burger	7

For the expressiveness feature module, each rule is applied according to the order in FeatExp. For each axiom in the ontology, lines 4-9 of the algorithm are executed, therefore each rule is applied as follows. Applying R1 results in the removal of Axiom 4, R2 removes Axioms 24-29, R3 removes Axiom 23 and R4 removes Axiom 22, and then the assertions in axioms 31-36 are removed (R5), and finally those for equivalence and equality (R6, Axioms 2, 9, and 30) and complex classes (R7), being Axioms 11 and 15.

IMPLEMENTATION AND EVALUATION OF THE ALGORITHMS

To solve the problem of laborious manual modularisation, we have created the tool NOMSA to modularise ontologies, which incorporates the abstraction and expressiveness algorithms presented in Section 4. NOMSA allows the user to upload an ontology (including its imports), and select one of five approaches to modularise it. A module is then generated. NOMSA is a stand-alone Java application and can be downloaded from `https://www.dropbox.com/sh/7a1ohvrn705upre/AACcqlFc3Ycd8kcYVTB4OTsFa?dl=0` together with its screencast demonstrating usage and the test files.

We have conducted a performance evaluation against other techniques that create other types of modules and it compared favourably to them, or at least in a similar timeframe. More interesting in the current scope is whether the new modules are any 'good' or 'useful'. For instance, if a certain type of abstraction always generates a module that is about the same size as the original, this is neither a good way of abstracting to generate a module nor useful for the ontology developer; vv., it is. Module quality is difficult to assess, however, and the current evaluation is, to the best of our knowledge, the first attempt to assess it quantitatively.

The selected metrics all have been proposed elsewhere and have been implemented in the TOMM tool that will be used for automated evaluation. Its metrics are summarized here so as to keep the chapter self-contained; see the TOMM documentation (Khan & Keet, 2016b) for longer descriptions and references where they have been proposed first:

- Size is the number of entities in a module |M|, counting classes, object and data properties, and individuals;
- Relative size of the module compared to the original ontology;
- Atomic size is the average size of a group of inter-dependent axioms in a module;
- Appropriateness is measured by mapping the size of an ontology module to some appropriateness function value between 0 and 1 where a module with an optimal size is of value 1;
- Intra-module distance is the distance between all entities in a module by counting the number of relations in the shortest path from one entity to another
- Relative intra-module distance is the difference between distances of entities in a module M to a source ontology O;
- Cohesion refers to the extent to which entities in a module are related to each other;
- Attribute richness is the average number of attributes per class;
- Inheritance richness describes how the knowledge is distributed across the ontology and is defined as the number of subclasses per class in an ontology;
- Correctness states that every axiom that exists in the module also exists in the original ontology; nothing new should be added to the module; and
- Completeness holds if the meaning of every entity is preserved as in the source ontology.

The method for the experiment is as follows:

1. Take a set of ontologies;
2. Run each modularisation tool's algorithm with a subset of 10 randomly selected ontologies from the test files to compare its features to NOMSA;

3. Run the NOMSA tool for each ontology from the test files for all five algorithms;
4. Run the Tool for Ontology Module Metrics (TOMM) (Khan & Keet, 2016b) for NOMSA's modules to acquire metrics; and
5. Conduct an analysis from the metrics for each module.

The materials used for the experiment were as follows: Protégé v4.3 (Musen, 2015), SWOOP (Kalyanpur et al., 2006) , OWL Module Extractor (Grau et al., 2008),PROMPT (Natalya Fridman Noy & Musen, 2004), PATO (Stuckenschmidt & Schlicht, 2009), TaxoPart (Hamdi et al., 2010), NOMSA using the default parameters, TOMM (Khan & Keet, 2016b), and 128 ontologies experimentation (Gardiner et al., 2006; Lawrynowicz & Keet, 2016). The default parameters for each algorithm in NOMSA are as follows: element type = object properties in VocAbs, level = 3 in HLAbs, and relative threshold = 50% value in WeiAbs. The set of 128 ontologies were from various domains, and derived from the set of ontologies described elsewhere for experimentation (Gardiner et al., 2006; Lawrynowicz & Keet, 2016).The ontologies in the data set ranged in size from with 0 to 10 520 number of entities. Our tests were carried out on a 3.00 GHz Intel Core 2 Duo PC with 4 GB of memory running Windows 7 Enterprise.

RESULTS

The results of comparing NOMSA's features to the existing modularisation tools are shown in Table 3. For most of the features, NOMSA performs as well as or better than the other tools, with the benefit of full automation of the process. For the level of interaction, NOMSA is automatic; it is possible to run NOMSA without the user providing any initial input parameters. NOMSA includes the greatest number of algorithms in a tool (five) compared to the other tools. Each tool was classified by techniques from the existing framework for modularisation (Khan & Keet, 2015). For techniques, NOMSA used semantic-based abstraction and language simplification techniques; semantic-based abstraction has not been applied in other tools to-date. NOMSA's algorithms take between 2-4 seconds to modularise. The locality-based algorithms have the quickest time (1 second) while partitioning algorithms take longer (6-16 seconds). It was not possible to test PROMPT for time as it was completely user-driven. We do not compare the resultant modules of the other modularisation tools because they all generate different types of modules and their underlying techniques differ.

All 128 ontologies were successfully modularised using all five algorithms in NOMSA and their metrics were generated using the TOMM module metrics tool (Khan & Keet, 2016b). The numerical metrics for the modules are displayed in Table 4 and we discuss the notable metrics here. For the data set in use, all five algorithms result in a reduction of the size of the original ontology, ranging from modules that have a relative size of 0.26 (WeiAbs) to modules that have a size of 0.85 (VocAbs) meaning that WeiAbs and VocAbs results in modules that are 26% and 85% the size of the original ontology, respectively. For the relative intra module distance, the modules of HLAbs and WeiAbs have values greater than 1 (18.66 and 3.96 respectively) meaning that the entities in the module are to that degree closer than in the original ontology; this could aid in human comprehension of an ontology. The rest of the algorithms have values less than 1, meaning that the entities are to that degree further away than in the original ontology. For instance, removing axioms that describe equality between classes simplifies the expressivity of the module for easier tool processing, but increases the distance between classes.

Table 3: Comparison of three features of the main modularisation tools against NOMSA and the average running times of the respective algorithms for

the test set of ontologies (excluding the time of manual modularisation tasks of the other tools, such as loading the ontology and setting the parameters)

	Level of interaction	**Algorithm complexity**	**Technique**	**Time (s.)**
SWOOP algorithm 1	Semi-automatic	Quadratic	Locality	1
SWOOP algorithm 2	Automatic	Quadratic	Graph partition	6
OWL module extractor	Semi-automatic	Quadratic	Locality	1
PROMPT	User driven	Unknown	Query	-
PATO	Automatic	Unknown	Graph partition	16
Protégé algorithm 1	Semi-automatic	Unknown	Locality	1
Protégé algorithm 2	Automatic	Unknown	Language based	1
Protégé algorithm 3	Semi-automatic	Unknown	Locality	1
Protégé algorithm 4	Semi-automatic	Unknown	Language based	1
TaxoPart	Automatic	Linear	Graph partition	15
NOMSA AxAbs	Automatic	Linear	Semantic based abstraction	3
NOMSA VocABs	Automatic	Linear	Semantic based abstraction	2
NOMSA HLAbs	Automatic	Quadratic	Semantic based abstraction	2
NOMSA WeiAbs	Automatic	Linear	Semantic based abstraction	4
NOMSA FeatExp	Automatic	Quadratic	Language based	3

In order to determine whether the modules are of good quality, we can compare the results obtained from the generated modules, to what is expected (we refer to the benchmark dependencies between modularity metrics of the framework for ontology modularity (Khan & Keet, 2016b). Comparing the modules to the dependencies, for WeiAbs and FeatExp modules, all the metric values for the generated modules correspond with what is expected; these modules are of 'good' quality. For AxAbs, VocAbs, and HLAbs, some of the metrics do not correspond to the dependencies. For the 128 modules, AxAbs and HLAbs succeeds to meet 1 out of the 2 expected values for the modules. WeiAbs succeeds to meet 2 out of the 2 expected values for the modules and FeatExp succeeds to meet the 1 expected value for the modules.

Table 4: The average values for the metadata for all the generated modules; app. = appropriateness, IMD =intra module distance, coh. = cohesion, AR = attribute richness, IR = inheritance richness, rel. = relative, T(s.)= time in seconds. The metrics that count toward 'good' quality modules are in bold font, the remainder are 'average'

	Size	**Atomic**	**App.**	**IMD**	**Coh.**	**AR**	**IR**	**Rel.**	**Rel.**	**T(s.)**

		Size						Size	IMD	
AxAbs Modules	238.04	2.34	0.19	866345.6	**0.06**	0.49	4.84	0.71	0.68	3.83
VocAbs Modules	443.38	3.24	0.19	848372.2	**0.06**	0.45	4.78	0.85	0.79	2.47
HLAbs Modules	202.77	3.48	0.24	166797.1	**0.03**	0.47	4.86	0.67	18.66	2.40
WeiAbs Modules	138.58	3.40	0.30	142698.4	**0.07**	0.39	2.72	**0.26**	3.96	3.53
FeatExp Modules	291.89	2.44	0.18	757305.1	**0.06**	0.25	4.80	0.72	0.70	2.83
Original	464.67	3.80	0.15	1866430	0.04	1.04	4.78	-	-	-

We now examine why some of the metrics fail for the algorithms. VocAbs and HLAbs modules are expected to have a large appropriateness value (>0.75) according to the benchmark dependencies. However, based on the calculation of the appropriateness value (Khan & Keet, 2016b), this is only possible where a module has between 167-333 axioms and in some cases a source ontology may have fewer axioms than 167; hence, the source ontology was already not in range and modularising it causes a further decrease in the axioms. For the ontologies of this experiment, for 50 of the 128 source ontologies, the number of axioms were less than 167, which would not result in the optimal appropriateness values. Following these finding, it appears that the appropriateness value needs to be potentially omitted from the dependency diagram for expected metrics. For AxAbs, the correctness measure needs to be true, but it fails meaning that some 'new' axioms were added to the module. Upon examining the TOMM metrics log files, it was found that, by the algorithm removing object properties, other entities have to be changed to be represented in the module. For instance, in the source Pizza ontology, 'France' is represented as using an object property, as follows: `MozarellaTopping` $\sqsubseteq$ $\exists$`hasCountryOfOrigin.{Italy}` and `{Italy}` $\neq$ `{France}` whereas, in the module using the AxAbs algorithm, 'France' is represented using a named individual, as follows: `{France}: Country`. TOMM recognizes this axiom as a new axiom in the module.

DISCUSSION

We have inventarised and structured notions of abstraction in the context of ontologies and related artefacts, which encompasses all the known notions of abstraction from the literature. Several categories from the theory were selected to use for automating modularisation ideas that were identified elsewhere (Khan & Keet, 2015) and hitherto could be carried out only manually. Their automation opens the road to examine quantitatively whether such methods are actually effective or not as process of abstraction. Given the results presented in the previous section, it is worthwhile to investigate the success of automating the remaining types of abstraction listed in Section 3.

The algorithms and tool designed for abstraction solve the problems of users' reliance on manual methods for modularity and the lack of abstraction techniques in existing tools. Our experiments show that our algorithms can be used to automatically modularise ontologies thus, it both broadens the scope of the extant set of algorithms for automated modularisation (Grau et al., 2008; Hamdi et al., 2010; Kalyanpur et al., 2006; Musen, 2015; Natalya Fridman Noy & Musen, 2004; Stuckenschmidt & Schlicht, 2009) to enable generation of more types of modules, and it refines and realises theory-based approaches, such as presented in (Ghidini & Giunchiglia, 2004; Keet, 2005, 2007; Mani, 1998; Pandurang Nayak & Y. Levy, 1995) so that it is usable by ontology engineers.

The performance for the algorithms is good; the time taken to modularise the ontologies is fast for all five algorithms (under 5 seconds on average for each ontology). Assessing the quality of the metrics of the modules reveal that for this test set of ontologies, WeiAbs and FeatExp algorithms generate modules of 'good' quality for all its modules according to the expected dependencies. For the remaining three algorithms, they generate some 'good' quality modules, but it is not possible to meet the expected metric values for all the modules; for some of the metrics depend on the source ontology. The resulting modules, are, however, still an improvement compared to the original ontologies; the sizes of the modules have been reduced considerably, and other metrics such as attribute richness, etc., are notably different when compared to the original ontologies, as displayed in Table 2.

The use of existing module metrics uncovered interesting results. The metrics, and the tool that implements them, TOMM, were evaluated on a set of 189 existing ontology modules (Khan & Keet, 2016b). This is slightly different from evaluating the process of generating modules. One may, however, extrapolate from the quality of a module to the process it has created: if the module is not good, then either neither was the process good or neither was the source ontology good, or both. For instance, instead of Schlicht and Stuckenschmidt's appropriateness value having been informed by software modules (Schlicht & Stuckenschmidt, 2006) (and integrated in TOMM as such), this could also be judged or specified upfront by the user or by type of ontology or by more data on the size of current ontologies and modules. Regarding the latter, the quality of the ontology that will be subjected to some abstraction operation: if it has representation issues then the resultant smaller ontology may also have them; OWL's enumerations/one-of—i.e., pretending that an instance is a class (universal/concept/type)—is one such modelling construct that is subject to debate.

CONCLUSION

Fourteen types of abstractions were identified and categorised, which may assist with the understanding of abstraction for achieving ontology modularisation. For five of them, new algorithms were designed to generate abstraction-based modules. They have been implemented in the NOMSA tool to modularise ontologies accordingly, which are the first fully-automated abstractions for ontologies. The quantitative evaluation of the modules' quality with TOMM showed that for the weighted abstractions (WeiAbs) and feature expressiveness (FeatExp) algorithms, the modules match the expected values from the framework for modularity for 'good' quality modules. For the axiom (AxAbs), vocabulary (VocAbs), and high level (HLAbs) abstraction algorithms, some of its modules match the expected values and some fail to match all the values due to the source ontology. All the generated modules, however, are notably different from the source ontologies according to their metrics.

For future work, it is worthwhile to investigate the design and implementation of algorithms and formalisations for the other categories of abstraction structured in Section 3 for ontology modularisation and to examine further the notion of 'good' abstractions and module metrics.

REFERENCES

Amato, F., De Santo, A., Moscato, V., Persia, F., Picariello, A., & Poccia, S. R. (2015). Partitioning of ontologies driven by a structure-based approach. *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*, 320–323.

Beisswanger, E., Schulz, S., Stenzhorn, H., & Hahn, U. (2008). BioTop: An upper domain ontology for the life sciencesA description of its current structure, contents and interfaces to OBO ontologies. *Applied Ontology*, *3*(4), 205–212. https://doi.org/10.3233/AO-2008-0057

Botoeva, E., Calvanese, D., & Rodriguez-Muro, M. (2010). *Expressive Approximations in DL-Lite Ontologies BT - Artificial Intelligence: Methodology, Systems, and Applications* (D. Dicheva & D. Dochev (eds.); pp. 21–31). Springer Berlin Heidelberg.

Campbell, L. J., Halpin, T. A., & Proper, H. A. (1996). Conceptual schemas with abstractions making flat conceptual schemas more comprehensible. *Data & Knowledge Engineering*, *20*(1), 39–85. https://doi.org/https://doi.org/10.1016/0169-023X(96)00005-5

Chen, J., Alghamdi, G., Schmidt, R. A., Walther, D., & Gao, Y. (2019). *Ontology Extraction for Large Ontologies via Modularity and Forgetting. BT - Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019.* (pp. 45–52). https://doi.org/10.1145/3360901.3364424

Cowell, L., & Smith, B. (2009). Infectious Disease Ontology. In *Infectious Disease Informatics* (pp. 373–395). https://doi.org/10.1007/978-1-4419-1327-2_19

d'Aquin, M., Sabou, M., & Motta, E. (2006). *Modularization: a key for the dynamic selection of relevant knowledge components*.

de Lara, J., Guerra, E., & Sánchez Cuadrado, J. (2013). Reusable abstractions for modeling languages. *Information Systems*, *38*(8), 1128–1149. https://doi.org/https://doi.org/10.1016/j.is.2013.06.001

Del Vescovo, C. (2011). The modular structure of an ontology: Atomic decomposition towards applications. *24th International Workshop on Description Logics*, 466.

Del Vescovo, C., Gessler, D. D. G., Klinov, P., Parsia, B., Sattler, U., Schneider, T., & Winget, A. (2011). *Decomposition and Modular Structure of BioPortal Ontologies BT - The Semantic Web – ISWC 2011* (L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, & E. Blomqvist (eds.); pp. 130–145). Springer Berlin Heidelberg.

Donnelly, K. (2006). SNOMED-CT: The advanced terminology and coding system for eHealth. *Studies in Health Technology and Informatics*, *121*, 279.

Federhen, S. (2012). The NCBI Taxonomy database. *Nucleic Acids Research*, *40*(Database issue), D136–D143. https://doi.org/10.1093/nar/gkr1178

Fridman, L., Stolerman, A., Acharya, S., Brennan, P., Juola, P., Greenstadt, R., & Kam, M. (2015). Multi-modal decision fusion for continuous authentication. *Computers & Electrical Engineering*, *41*, 142–156.

Gardiner, T., Tsarkov, D., & Horrocks, I. (2006). *Framework for an Automated Comparison of Description Logic Reasoners BT - The Semantic Web - ISWC 2006* (I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, & L. M. Aroyo (eds.); pp. 654–667). Springer Berlin Heidelberg.

Ghidini, C., & Giunchiglia, F. (2004). *A semantics for abstraction*.
Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2008). Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, *31*, 273–318.
Guarino, N. (1998). Formal Ontology and Information Systems. In N. Guarino (Ed.), *Proceedings of the 1st International Conference on Formal Ontology in Information Systems (FOIS'98)* (pp. 3–15). IOS Press.
Hamdi, F., Safar, B., Reynaud, C., & Zargayouna, H. (2010). *Alignment-Based Partitioning of Large-Scale Ontologies BT - Advances in Knowledge Discovery and Management* (F. Guillet, G. Ritschard, D. A. Zighed, & H. Briand (eds.); pp. 251–269). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00580-0_15
Jaeschke, P., Oberweis, A., & Stucky, W. (1994). *Extending ER model clustering by relationship clustering BT - Entity-Relationship Approach — ER '93* (R. A. Elmasri, V. Kouramajian, & B. Thalheim (eds.); pp. 451–462). Springer Berlin Heidelberg.
Janowicz, K., & Compton, M. (2010). The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration into the Semantic Sensor Network Ontology. *SSN*.
Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., & Hendler, J. (2006). Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics*, *4*(2), 144–153. https://doi.org/https://doi.org/10.1016/j.websem.2005.10.001
Keet, C. M. (2007). *Enhancing Comprehension of Ontologies and Conceptual Models Through Abstractions BT - AI*IA 2007: Artificial Intelligence and Human-Oriented Computing* (R. Basili & M. T. Pazienza (eds.); pp. 813–821). Springer Berlin Heidelberg.
Keet, C. M. (2005). *Using Abstractions to Facilitate Management of Large ORM Models and Ontologies BT - On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops* (R. Meersman, Z. Tari, & P. Herrero (eds.); pp. 603–612). Springer Berlin Heidelberg.
Khan, Z. C., & Keet, C. M. (2015). An empirically-based framework for ontology modularisation. *Applied Ontology*, *10*(3–4). https://doi.org/10.3233/AO-150151
Khan, Z. C., & Keet, C. M. (2016a). ROMULUS: The Repository of Ontologies for MULtiple USes Populated with Mediated Foundational Ontologies. *Journal on Data Semantics*, *5*(1). https://doi.org/10.1007/s13740-015-0052-1
Khan, Z. C., & Keet, C. M. (2016b). *Dependencies Between Modularity Metrics Towards Improved Modules BT - Knowledge Engineering and Knowledge Management* (E. Blomqvist, P. Ciancarini, F. Poggi, & F. Vitali (eds.); pp. 400–415). Springer International Publishing.
Krötzsch, M. (2012). *OWL 2 Profiles: An Introduction to Lightweight Ontology Languages BT - Reasoning Web. Semantic Technologies for Advanced Query Answering: 8th International Summer School 2012, Vienna, Austria, September 3-8, 2012. Proceedings* (T. Eiter & T. Krennwallner (eds.); pp. 112–183). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33158-9_4
Lawrynowicz, A., & Keet, C. M. (2016). *The TDDonto Tool for Test-Driven Development of DL Knowledge bases. BT - Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.* http://ceur-ws.org/Vol-1577/paper_15.pdf
LeClair, A., Khédri, R., & Marinache, A. (2019). *Toward Measuring Knowledge Loss due to Ontology Modularization. BT - Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2019, Volume 2: KEOD, Vienna, Austria, September 17-19, 2019.* (pp. 174–184).

https://doi.org/10.5220/0008169301740184
Mani, I. (1998). *A Theory of Granularity and its Application to Problems of Polysemy and Underspecification of Meaning.*
Mikroyannidi, E., Rector, A., & Stevens, R. (2009). Abstracting and generalising the foundational model anatomy (fma) ontology. *Proceedings of the Bio-Ontologies 2009 Conference*.
Mossakowski, T., Codescu, M., Neuhaus, F., & Kutz, O. (2015). *The Distributed Ontology, Modeling and Specification Language – DOL BT - The Road to Universal Logic: Festschrift for the 50th Birthday of Jean-Yves Béziau Volume II* (A. Koslow & A. Buchsbaum (eds.); pp. 489–520). Springer International Publishing. https://doi.org/10.1007/978-3-319-15368-1_21
Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI Matters*, *1*(4), 4–12.
Newman, D., Bechhofer, S., & De Roure, D. (2009). *myExperiment: An ontology for e-Research*.
Noy, Natalya F, & Musen, M. A. (2009). *Traversing Ontologies to Extract Views BT - Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization* (H. Stuckenschmidt, C. Parent, & S. Spaccapietra (eds.); pp. 245–260). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01907-4_11
Noy, Natalya Fridman, & Musen, M. A. (2004). *Specifying Ontology Views by Traversal. BT - The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings* (pp. 713–725). https://doi.org/10.1007/978-3-540-30475-3_49
Noy, Natalya Fridman, & Musen, M. A. (2000). {PROMPT}: Algorithm and Tool for Automated Ontology Merging and Alignment. *Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 450–455.
Pandurang Nayak, P., & Y. Levy, A. (1995). A Semantic Theory of Abstractions. In *Proceedings of IJCAI-95*.
Rosse, C., & Mejino, J. L. V. (2003). A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, *36*(6), 478–500. https://doi.org/https://doi.org/10.1016/j.jbi.2003.11.007
Schlicht, A., & Stuckenschmidt, H. (2006). *Towards Structural Criteria for Ontology Modularization. BT - Proceedings of the 1st International Workshop on Modular Ontologies, WoMO'06, co-located with the International Semantic Web Conference, ISWC'06 November 5, 2006, Athens, Georgia, USA*. http://ceur-ws.org/Vol-232/paper7.pdf
Schulz, S., & Boeker, M. (2013). *BioTopLite: An Upper Level Ontology for the Life SciencesEvolution, Design and Application. BT - Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt, 16.-20. September 2013,* (pp. 1889–1899). https://dl.gi.de/20.500.12116/20620
Stuckenschmidt, H., & Schlicht, A. (2009). *Structure-Based Partitioning of Large Ontologies. BT - Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization* (pp. 187–210). https://doi.org/10.1007/978-3-642-01907-4_9
Studer, T. (2010). *Privacy Preserving Modules for Ontologies BT - Perspectives of Systems Informatics* (A. Pnueli, I. Virbitskaite, & A. Voronkov (eds.); pp. 380–387). Springer Berlin Heidelberg.