NOMSA: Automated modularisation for abstraction modules

Zubeida Casmod Khan[1] and C. Maria Keet[2]

[1] Council for Scientific and Industrial Research, Pretoria, South Africa
zkhan@csir.co.za
[2] Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za

Abstract. Large and complex ontologies lead to difficulty in usage by humans and causes processing problems with software agents. Modularity has been proposed to address this problem. Current methods and tools can be used to create only some of the existing types of required modules. To augment options for modularisation, we present novel methods to create five types of abstraction modules: axiom abstraction, vocabulary abstraction, high-level abstraction, weighted abstraction, and feature expressiveness. They have been implemented in the novel tool NOMSA for automated modularisation, which also offers a GUI.

1 Introduction

Ontologies that describe large, well-defined domains are consequently large and complex in nature, such as the FMA [7] and SNOMED CT [5]. Current software tools face inherent difficulties to process such ontologies due to computational limitations while humans face cognitive overload in trying to understand them. Therefore, the need for modularisation of ontologies is becoming more prevalent as it may offer benefits such as carving up the knowledge into manageable chunks for humans and tools alike.

There has been an increase in methods and tools to assist with such a modularisation process [1, 6, 2]. While there are many modularisation tools, most types of modules are still manually created. One such type is abstraction modules. Abstraction is the principle of simplifying complex models by removing 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. We have investigated methods for creating abstraction modules, towards creating new algorithms for them. These algorithms have been implemented in a GUI called NOMSA (Novel Ontology Modularisation SoftwAre).

Section 2 briefly describes the methods, which are illustrated with a sample ontology. Details of the implementation are described in Section 3.

2 Modularisation methods

We focus on designing algorithms to match those modularisation options that do not have any means of realising them, as was observed in an existing evidence-

based ontology modularisation framework [4]. In particular, there are five types of abstraction modules that are fall short with respect to algorithms and implementations, which we summarise here and are illustrated afterward.

- *Axiom abstraction* Axiom abstraction generates a module without complex relations between classes; therefore, the technique decreases the mesh structure of the ontology (if present) and makes it a 'bare' taxonomy of classes and unused object properties.
- *Vocabulary abstraction* Applying this abstraction to an ontology generates a module where a certain vocabulary element is removed from the ontology, or a whole group of elements is removed (e.g., all data types, if present).
- *High-level abstraction* generates a module where entities at a higher level in the hierarchy are regarded more important than others. This introduces the notion of desired *depth* to specify in the abstraction process.
- *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. We determine importance by assessing 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. Of course, what is used 'often' and what is 'less important' may be relative and thus depend on the ontology. Therefore, the weighted abstraction includes a user-definable *threshold*, which may be absolute or relative.
- *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. We have designed 7 rules for this. The algorithm takes these 7 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. We summarise the seven rules here, with the type of axioms that are to be removed. *R1: Qualified cardinality* deals with cardinality constraints between classes and properties. *R2: Domain and range* pertains to axioms that have been specified using domain and range for object properties. *R3: Object property characteristics* are those axioms pertaining to logical characteristics of object properties such as symmetry and transitivity. *R4: Disjointness* are those axioms pertaining to the disjointness of classes. *R5: Assertions* are those axioms that are assertions between individuals and classes, or properties. *R6: Atomic equivalence and equality* are those axioms that state equivalence between entities. *R7: Complex classes* are those axioms that contain intersection and union logical operators.

Consider the axioms in a toy Burger ontology in Fig. 1 (entity declaration axioms omitted). Running *axiom abstraction* on this ontology would remove

$$\begin{aligned}
&\mathsf{BeefPatty} \sqsubseteq \mathsf{Patty} && (1)\\
&\mathsf{Beefburger} \equiv \mathsf{HamBurger} && (2)\\
&\mathsf{Beefburger} \sqsubseteq \mathsf{Burger} && (3)\\
&\mathsf{Cheapburger} \sqsubseteq \leq 1\ \mathsf{hasFilling}.\mathsf{Filling} && (4)\\
&\mathsf{Cheapburger} \sqsubseteq \mathsf{Burger} && (5)\\
&\mathsf{Cheese} \sqsubseteq \mathsf{Filling} && (6)\\
&\mathsf{Chef} \sqsubseteq \mathsf{Person} && (7)\\
&\mathsf{Customer} \sqsubseteq \mathsf{Person} && (8)\\
&\mathsf{HamBurger} \sqsubseteq \mathsf{Burger} && (9)\\
&\mathsf{HealthyBurger} \sqsubseteq \forall \mathsf{hasFilling}.(\mathsf{Lettuce} \sqcup \mathsf{Tomato}) && (10)\\
&\mathsf{HealthyBurger} \sqsubseteq \mathsf{Burger} && (11)\\
&\mathsf{Lettuce} \sqsubseteq \mathsf{Filling} && (12)\\
&\mathsf{Medium} \sqsubseteq \mathsf{PattyCook} && (13)\\
&\mathsf{PattyCook} \equiv \mathsf{Medium} \sqcup \mathsf{Rare} \sqcup \mathsf{WellDone} && (14)\\
&\mathsf{Rare} \sqsubseteq \mathsf{PattyCook} && (15)\\
&\mathsf{Sauce} \sqsubseteq \mathsf{Filling} && (16)\\
&\mathsf{Tomato} \sqsubseteq \mathsf{Filling} && (17)\\
&\mathsf{WellDone} \sqsubseteq \mathsf{PattyCook} && (18)\\
&\mathsf{WhiteBun} \sqsubseteq \mathsf{BurgerBun} && (19)\\
&\mathsf{WholeWheatBun} \sqsubseteq \mathsf{BurgerBun} && (20)\\
&\mathsf{WholeWheatBun} \sqsubseteq \neg\mathsf{WhiteBun} && (21)\\
&\mathsf{Func}(\mathsf{hasBun}) && (22)\\
&\exists\ \mathsf{hasBun}.\top \sqsubseteq \mathsf{Burger} && (23)\\
&\top \sqsubseteq \forall \mathsf{hasBun}.\mathsf{BurgerBun} && (24)\\
&\exists\ \mathsf{hasPatty}.\top \sqsubseteq \mathsf{Burger} && (25)\\
&\top \sqsubseteq \forall \mathsf{hasPatty}.\mathsf{Patty} && (26)\\
&\exists\ \mathsf{hasPattyCook}.\top \sqsubseteq \mathsf{Patty} && (27)\\
&\top \sqsubseteq \forall \mathsf{hasPattyCook}.\mathsf{PattyCook} && (28)\\
&\mathsf{MarthasBurger} \not\equiv \mathsf{MyBurger} && (29)\\
&\mathsf{MarthasBurger} : \mathsf{Burger} && (30)\\
&\mathsf{MyBurger} : \mathsf{Beefburger} && (31)\\
&\mathsf{MyBurger} : \mathsf{Burger} && (32)\\
&\mathsf{ChefRose} : \mathsf{Chef} && (33)\\
&\mathsf{cookedBy}(\mathsf{MyBurger}, \mathsf{ChefRose}) && (34)
\end{aligned}$$

Fig. 1. The burger ontology to which the algorithms are applied; see text for details.

the axioms numbered 4, 10, and 24-28, for they involve object properties of classes. For *vocabulary abstraction* there are several options. Let us remove all the instances, as most ontologies focus on the TBox anyway. This would remove the axioms numbered 30-33 and as knock-on effect, also axiom numbers 29 and 36. The *high level abstraction* is, perhaps, not of much interest in this sample ontology, for there are few hierarchies. Setting the depth at level 1 is the only way to actually have something removed, as there are only two levels. This would remove all the types of burgers, of buns, and of fillings, i.e., the axioms numbered 2, 3, 5-13, and 15-21.

To generate a *weighted abstraction* module, 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%. The threshold value represents an amount of the ontology that is to be removed. First we weigh each class in the ontology with its number of referencing axiom. Thereafter, 25% of the classes with the lowest values are removed (amounting to 5), as displayed in Table 1. That is, the classes that are deemed least important are those with the lowest number of referencing axioms; e.g., WhiteBun is only referred to in axioms 19 and 21, whereas Filling is referred to in axioms 4, 6, 12, 16, 17, hence, WhiteBun is removed. We omit the feature abstraction illustration due to space limitations.

3 Implementation of NOMSA

To solve the problem of manual modularisation, we have created the tool NOMSA to modularise ontologies, which incorporates the five abstraction algorithms. NOMSA allows the user to upload an ontology (including any imports), and select an approach to modularise it. Each approach is satisfied by a novel algorithm which correspond to the abstractions described in Sect. 2. The algorithms are available as online supplementary material (`http://www.thezfiles.co.za/`

Table 1. The classes of the Burger ontology with the number of referencing axioms. For the weighted abstraction module, those in bold font are the classes to be removed.

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
WholeWheat Bun	**2**	WellDone	3	PattyCook	6
Person	3	Rare	3	Burger	7

`modularisation`). NOMSA is a stand-alone application that can be downloaded from the aforementioned URL as well as a screencast.

We have evaluated the algorithms on 128 ontologies. All the generated modules are notably different from the source ontologies, as is the case also for the Burger example (see Table 2) whose metrics were generated with TOMM [3]. We are looking into refining and module quality metrics to compare modules.

Table 2. Selected metrics of the Burger WeiAbs module and original ontology.

	Size	**No. of axioms**	**Correctness**	**Completeness**
Burger (Original)	29	58	-	-
Burger (WeiAbs)	15	26	True	False
Burger (AxAbs)	29	44	True	False
Burger (HLAbs level 3)	26	52	True	True

References

1. Amato, F., Santo, A.D., Moscato, V., Persia, F., Picariello, A., Poccia, S.R.: Partitioning of ontologies driven by a structure-based approach. In: IEEE ICSC'15. pp. 320–323. IEEE (2015), anaheim, CA, USA, Feb 7-9, 2015
2. Kalyanpur, A., Parsia, B., Sirin, E., Cuenca Grau, B., Hendler, J.A.: Swoop: A Web Ontology Editing Browser. J. of Web Semantics 4(2), 144–153 (2006)
3. Khan, Z.C., Keet, C.M.: Dependencies between modularity metrics towards improved modules. In: Proc. of EKAW'16. LNAI, vol. 10024, pp. 400–415. Springer (2016)
4. Khan, Z.C., Keet, C.M.: An empirically-based framework for ontology modularisation. Applied Ontology 10(3-4), 171–195 (2015)
5. Lee, D., Cornet, R., Lau, F., de Keizer, N.: A survey of SNOMED CT implementations. J. of Biomedical Informatics 46(1), 87 – 96 (2013)
6. Romero, A.A., Kaminski, M., Grau, B.C., Horrocks, I.: Module extraction in expressive ontology languages via datalog reasoning. J. of Artificial Intelligence Research 55, 499–564 (2016)
7. Rosse, C., Mejino Jr, J.L.V.: A reference ontology for biomedical informatics: the foundational model of anatomy. J. of Biomedical Informatics 36(6), 478–500 (2003)