Mechanisms for Importing Modules[⋆]

Bijan Parsia, Ulrike Sattler, and Thomas Schneider

University of Manchester, UK {bparsia,sattler,schneider}@cs.man.ac.uk

Abstract The current OWL 2 specification provides mechanisms for importing whole ontologies. This paper discusses the import of only a module of an external ontology, which is specified by a set of terms (classes and properties) and defined in such a way that it contains "all knowledge" of the external ontology about these terms. We discuss possible design choices for such an import mechanism, based on the well-understood properties of logic-based module extraction techniques.

1 Introduction

When developing an ontology, it is helpful for the engineer if she can reuse information from external, already existing, ontologies. For example, when modelling knowledge related to a specific medical topic, it is useful to reuse knowledge from well-established medical ontologies such as Snomed CT [13], NCI[1] [5], FMA,[2] Galen,[3] and GO.[4][5] This means that the engineer, who is not necessarily an expert in all the fields covered by those ontologies, does not have to "reinvent" the representation of the knowledge therein. For the purpose of reusing ontologies, OWL 2 provides the `directlyImportsDocuments` association, from which the associations `directlyImports` and `imports` are derived.

The size of some existing ontologies such as Snomed CT or FMA makes it difficult for current tools to load and navigate through them on a standard computer, not to mention to classify them. It is therefore more economic to reuse parts of existing ontologies; ideally these parts *cover* all the knowledge about the terms the ontology engineer is interested in. There are many approaches to module extraction, see [14], which differ in their ability to provide coverage.

In order to make the notion of coverage more precise, let us assume that an ontology $\mathcal{O}$ directly imports an external ontology $\mathcal{E}$. Furthermore, assume that, in $\mathcal{O}$, all classes and properties used from the axiom closure of $\mathcal{E}$ are in our "interface signature" S, which is the set of terms that we are interested in reusing. Now assume that, instead of importing the whole ontology $\mathcal{E}$ as described in [11,

⋆ This work has been supported in part by the UK EPSRC grant no. EP/E065155/1.

[1] Latest version: `ftp://ftp1.nci.nih.gov/pub/cacore/EVS/NCI_Thesaurus`

[2] `http://sig.biostr.washington.edu/projects/fm/AboutFM.html`

[3] `http://www.co-ode.org/galen/`

[4] `http://www.geneontology.org/`

[5] Online browsers for some of these ontologies: `http://bioportal.nci.nih.gov/ncbo/faces/pages/ontology_list.xhtml`

Section 3.4], we only added a (hopefully much smaller) subset $M(S, \mathcal{E})$ of $\mathcal{E}$ to $\mathcal{O}$. A minimal condition for $M(S, \mathcal{E})$ being an acceptable substitute for $\mathcal{E}$ is that *$M(S, \mathcal{E})$ covers all knowledge that $\mathcal{E}$ has about S*. In other words, regardless of how $\mathcal{O}$ looks like and what it says about terms from S, if $\mathcal{O} \cup \mathcal{E}$ entails something (e.g., that 'john' is an instance of 'happy and man' or that 'neoplasm is a subclass of carcinoma'), then so does $\mathcal{O} \cup M(S, \mathcal{E})$. As a consequence of this coverage, we wouldn't notice the difference between importing $\mathcal{E}$ or $M(S, \mathcal{E})$—other than in the (hopefully much smaller) size of the resulting ontology, and as long as we use, in $\mathcal{O}$, only those terms from $\mathcal{E}$ that are in S.

Coverage is provided by only a few module notions, such as those based on conservative extensions [4, 10], E-connections [2], and locality [1, 6]. While modules based on conservative extensions are minimal coverage-providing modules, they cannot be extracted efficiently for OWL DL and even sublanguages thereof [4, 10], except for fragments of OWL EL [8] and OWL QL [15]. As a remedy, modules based on locality have been introduced, which sacrifice minimality for the efficiency of computing. We have shown recently that these modules are a good compromise [12], and have implemented their extraction in the OWL API.[6]

Since the current import mechanism of OWL only allows to import whole ontologies, we are suggesting a refinement that enables the import of coverage-providing modules. This refinement consists in adding the interface signature S as a parameter to the import statement and leaving the choice of a suitable module notion to the implementation level. We find this distribution over the specification and implementation level appropriate, for the following reasons. First, without allowing to specify the interface signature *at the language level*, we would not be able to provide modular import at all. If a tool encounters an unrestricted import statement, there is no way of deciding whether a module would be appropriate and, if so, which. Second, if we did not leave the choice of module type *to the implementation level*, we would have to fix a module type, which is hard given the above described circumstances: minimal coverage-providing modules for an OWL 2 ontology are computationally very difficult to extract, and there currently exists *one* good approximation that is efficient and applicable to OWL 2. If we committed the specification to the latter type of module, we would lose the chance to employ other approximations that might be developed, and shown to be more economic, in the future. Currently, if we assume that the tool makes a *reasonable* choice, there is not much of a choice to make given the small number of module extraction approaches that provide strong logical guarantees and efficient extraction algorithms.

In contrast to similar approaches to working with modules based on interfaces, such as [3], our approach does not require any features of the underlying languages. In particular, nominals are not required, but may be present; and no epistemic reasoning needs to be performed. Therefore, reasoning over the importing ontology is as hard as for the OWL 2 fragment determined by its import closure. We are not concerned with an extension of the underlying description logic; we only propose an extension of the structural specification of OWL 2.

[6] `http://owlapi.sourceforge.net`

We will discuss possible design choices and implementation issues, and briefly look at consequences of a refined import statement for collaborative ontology development.

2 Preliminaries

We consider an OWL DL ontology as a set of axioms and a module as a subset thereof. Since the module notions that are of interest focus on the *logical* axioms of an ontology, a module in the strict sense will not contain any non-logical axioms such as declarations or annotations. However, it is straightforward to retrospectively enrich the module of an ontology with all declarations and annotations for the terms that occur in it, and with all relevant ontology annotations.

We call ontologies to be reused *external* and denote them and their modules with $\mathcal{E}$ and $\mathcal{M}$, respectively. For ontologies importing $\mathcal{E}$ or $\mathcal{M}$, and for arbitrary ontologies, we use the denotation $\mathcal{O}$. An *interface signature* S is a set of class and property names. It acts as an interface for specifying the part of an ontology by listing the terms that are of interest for extracting a module. When we say that the subset $\mathcal{M}$ of $\mathcal{E}$ *covers all knowledge that* $\mathcal{E}$ *has about* S, we mean that, for every axiom α that uses terms only from S, we have that $\mathcal{E}$ entails α if and only if $\mathcal{M}$ entails α.[7] Then, and only then, we call $\mathcal{M}$ an *S-module* of $\mathcal{E}$.

Please note that the above definition of coverage is weaker than what we have described as coverage in Section 1. It will be made stronger below when we introduce robustness under replacement.

As pointed out in [12], there are stronger notions of a module that are of interest for certain applications. In order to introduce these, we call the set of terms (class, property and individual names) that occur in an ontology $\mathcal{O}$ the *signature* of $\mathcal{O}$, and denote it by $\mathrm{sig}(\mathcal{O})$. A subset $\mathcal{M}$ of $\mathcal{E}$ is called a *self-contained S-module of* $\mathcal{E}$ if $\mathcal{M}$ covers all knowledge that $\mathcal{E}$ has about $S \cup \mathrm{sig}(\mathcal{M})$. It is called a *depleting S-module of* $\mathcal{E}$ if the difference $\mathcal{E} \setminus \mathcal{M}$ has no knowledge about $S \cup \mathrm{sig}(\mathcal{M})$. For details see [12]. The first notion is stronger than the "plain" module notion in that it requires the module to preserve entailments that can be formulated in the interface signature *plus* the signature of the module. The second notion means that the difference of $\mathcal{E}$ and its module $\mathcal{M}$ does not entail any axioms in terms of $S \cup \mathrm{sig}(\mathcal{M})$ other than tautologies. It is not necessarily stronger than the first or the "plain" module notion; this depends on the expressivity of the underlying ontology language [12].

For reuse scenarios, it is desirable that the used module notion has certain robustness properties. Such robustness properties have been collected and examined in a more general context in [12]; here we are identifying them with their consequences on modular reuse, giving their implications, not the definitions.

- *Robustness under vocabulary restrictions* means that, if $\mathcal{M}$ is an S-module of $\mathcal{E}$, it is also an S'-module of $\mathcal{E}$, for any subset S' of S. This implies that, after restricting the interface signature, we can still use the old module.

[7] To be precise, "an ontology entails α" means that this ontology entails the ontology consisting of only the axiom α, with ontology entailment defined in [11, Section 2.5].

– *Robustness under vocabulary extensions* implies that, whenever $\mathcal{M}$ is an S-module of $\mathcal{E}$, it is also an S'-module of $\mathcal{E}$, for any set S' that does not extend S by any terms used in $\mathcal{E}$. This implies that, after extending the interface signature with terms irrelevant for $\mathcal{E}$, we do not have to import a new module.
– *Robustness under replacement* means that, whenever $\mathcal{M}$ is an S-module of $\mathcal{E}$ and $\mathcal{O}$ does not contain any terms outside S that are not in $\mathcal{E}$, $\mathcal{O} \cup \mathcal{M}$ is an S-module of $\mathcal{O} \cup \mathcal{E}$. This implies that coverage is preserved under imports into *arbitrary* OWL DL ontologies, and leads to the stronger notion of coverage we have described in Section 1 and which is essential for our import scenario where modules can be imported into arbitrary ontologies, which themselves can be imported in turn.
– *Robustness under joins* means that, if two ontologies cover the same knowledge about S without one necessarily being a subset of the other, it suffices to import one instead of both.

Not all coverage-providing module notions provide these properties, and it has turned out that certain locality-based modules are very robust in addition to being efficiently computable [12].

According to [11, Section 3.4], the *imports closure* of an ontology $\mathcal{O}$ is the set containing $\mathcal{O}$ and all the ontologies that $\mathcal{O}$ imports. The *axiom closure* of $\mathcal{O}$ is the smallest set that contains all the axioms from each ontology $\mathcal{O}'$ in the import closure of $\mathcal{O}$ with all anonymous individuals standardised apart, where the latter notion means that anonymous individuals from different ontologies in the import closure of $\mathcal{O}$ are treated as being different, see [11, Section 5.6.2].

3 An import mechanism for modules

In order to facilitate the import of modules, we propose a refined import association `directlyImportsDocuments` that can have two parameters: the interface signature as a list of class and property names, and the external ontology. The derived `directlyImports` association will have as a parameter a module of the external ontology for the interface signature, which has been extracted using a suitable module notion. The choice of module notion is discussed further below.

To be more precise, we propose a change to the Functional-Style Syntax in [11, Section 3.7], which consists in replacing the production

directlyImportsDocuments := { 'Import' '(' **IRI** ')' } with

```
directlyImportsDocuments := {
    'Import' '(' IRI ')' |
    'ImportModule' '(' SignatureTerm {SignatureTerm} IRI ')'   }

SignatureTerm :=
    'Class' '(' Class ')' |
    'ObjectProperty' '(' ObjectProperty ')' |
    'DataProperty' '(' DataProperty ')'
```

This will facilitate the import of a module of an ontology for the interface signature consisting of the specified class and property names.

We have chosen to allow no entities other than classes, object properties and data properties in the interface signature because of the module notion we prefer and advise. This is explained in more detail in the following.

Structural specification of `directlyImportsModule`. The refined import mechanism simply requires a preprocessing step where each `ImportModule` statement, followed by a list of terms and an ontology IRI, is replaced with a statement `Import`, followed by the IRI of the module computed for the terms and ontology of the given IRI. This requires a small change to the *canonical parsing* process specified in [11, Sec. 3.6], to be perform an additional step before CP 2.2:

> For each `ImportModule` statement in D_I, compute the corresponding module, store it in a new ontology document M with IRI I_M, and replace the `ImportModule` statement with an `Import` statement that has I_M as a parameter.

This ensures that the computed module will be treated as a new ontology document whose IRI occurs in the `directlyImportsDocuments` association. The derivation of the associations `directlyImports` and `imports` as given in [11, Section 3.4] remains unchanged. The same holds for the import closure and the axiom closure; the former will contain the new ontologies, and the latter will contain the axioms therein. No further changes are necessary to the parsing process or the structure of the associations.

It is worth noting that certain properties which held trivially for `Import` do not carry over to `ImportModule`, which might be counterintuitive at first sight. In particular, imports are not commutative or associative anymore. This means that, if we use "$\mathcal{O}_1$ imports $\mathcal{O}_2$" as a shortcut for the imports closure of $\mathcal{O}$ plus the statement `Import`$(\mathcal{O}_2)$, then the following properties hold, but do in general not carry over if `Import` is replaced by `ImportModule`.

$$\begin{aligned}
\mathcal{O}_1 \text{ imports } \mathcal{O}_2 &= \mathcal{O}_2 \text{ imports } \mathcal{O}_1 \\
(\mathcal{O}_1 \text{ imports } \mathcal{O}_2) \text{ imports } \mathcal{O}_3 &= \mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) \\
\mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) &= \mathcal{O}_1 \text{ imports } (\mathcal{O}_3 \text{ imports } \mathcal{O}_2)
\end{aligned}$$

Minimal requirements to the extraction of the module. In order to obtain correct results for cyclic imports and import chains, it is necessary to extract the module from the imports closure of the imported ontology. In the case of cyclic import, the imports closure of an ontology $\mathcal{O}$ contains all ontologies in all cycles involving $\mathcal{O}$. In the case of import chains, the order in which the ontologies are parsed matters. For instance, if there are ontologies $\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2$ with IRIs I_1, I_2, I_3 such that, for $i = 0, 1$, $\mathcal{O}_{i+1}$ contains a statement `ImportModule`(S_i, I_i) or `Import`(I_i). If the statement in $\mathcal{O}_0$ is an `ImportModule`-statement rather than an `Import`-statement, then it will be necessary to ensure that $\mathcal{O}_1$ is parsed before $\mathcal{O}_0$. This can always be achieved for arbitrary import graphs because we have just seen that cycles are unified with the union of the involved ontologies. We thus obtain a partial order in which the ontologies need to be parsed.

Furthermore, the extraction of the module will need to ensure that the following two restrictions to the import closure from [11, Section 3.4] are met:

"The import closure of $\mathcal{O}$ *should not* contain ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ such that
– $\mathcal{O}_1$ and $\mathcal{O}_2$ are different ontology versions from the same ontology series, or
– $\mathcal{O}_1$ contains an ontology annotation `owl:incompatibleWith` with the value equal to either the ontology IRI or the version IRI of $\mathcal{O}_2$."

This can be ensured by requiring that the module of an ontology always contains the relevant annotations to the ontology it has been extracted from. We will discuss this together with the choice of module type.

The choice of module type. In principle, the above specification does not restrict the way a module is extracted, except for the requirement that every module of an ontology $\mathcal{E}$ should inherit version and incompatibility annotations from $\mathcal{E}$. We do, however, strongly advise the use of module extraction approaches that provide coverage because, to the best of our knowledge, this is the only way to ensure that the knowledge about the specified terms is preserved. We furthermore suggest that modules should always be self-contained and depleting because these are properties that are relevant for some applications, see [12].

Another important requirement to the module notion is economy, which has two characteristics: (a) modules should be as small as possible because otherwise the full ontology could be reused, which would trivially ensure coverage; and (b) modules should be obtained efficiently. It is known that it is difficult to combine these two characteristics for the description logic that underlies OWL DL and many of its sublogics. For instance, there is no algorithm for extracting *minimal* coverage-providing modules for a given interface signature and OWL DL ontology because the underlying decision problem is undecidable [10].

Given the problems to combine (a) and (b), we believe, that a compromise between (a) and (b) is the most feasible solution. Such a compromise is provided by modules based on syntactic locality [1, 6], which approximate minimal coverage-providing modules. This means that, given an interface signature S and OWL DL ontology $\mathcal{E}$, a locality-based module (LBM) of $\mathcal{E}$ for S contains every minimal coverage-providing module of $\mathcal{E}$ for S, and therefore provides coverage as well. In general, LBMs can be larger, but initial experiments suggest that the difference in size does not have to be large [12]. Although in another set of experiments with OWL QL ontologies in [9], LBMs differed from minimal coverage-providing modules more significantly, this has to be relativised because the latter type of modules is usually neither self-contained nor depleting, but locality-based modules are. The size differences in [9] between LBMs and other, self-contained and depleting, modules are not as significant.

From the insights gained in [12], we recommend to use nested LBMs, namely $\top\bot^*$-modules. They have been implemented in the OWL API[8] and combine the following desirable properties.

[8] To be precise, the current release 2.2 only captures $\top$-, $\bot$-, $\top\bot$- and $\bot\top$-modules, but longer nesting chains can easily be obtained iteratively. The upcoming version 3 of the OWL API will contain $\top\bot^\star$-modules.

– They are efficiently computable.
– They are always self-contained and depleting modules.
– They have robustness properties that are relevant for reuse of ontologies, see Section 2. Please note that these properties do not, in general, hold for $\top\bot$- and $\bot\top$-modules, so it is essential to use the $*$-versions.

It is possible to make use of the few exceptions that, for some small fragments of OWL DL, smaller coverage-providing modules can be obtained efficiently. The most notable such case is probably that of acyclic OWL EL terminologies, for which the system MEX has been devised in [8, 7]. MEX modules have the same robustness properties as LBMs, provide coverage and are contained in the respective LBM. Therefore, if differences in module size really matter, it is possible to determine whether the current ontology falls into that fragment and, in the positive case, to extract a MEX module instead of an LBM. It is known that the problem whether a given ontology belongs to this fragment can be decided efficiently.

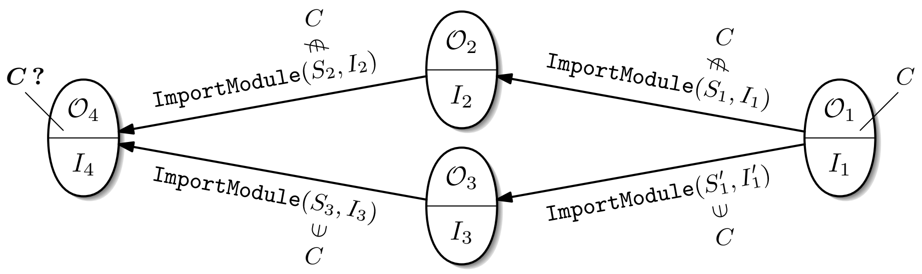
Since we strongly advise to use LBMs for importing modules, an interface signature cannot contain entities other than classes, object properties and data properties. This has to do with the way locality is defined; the reasons can roughly be summarised as follows. First, it is not possible, or too strong a restriction, to interpret an individual (or a datatype, respectively) as the empty set or the full domain (as the empty set or the full value space, respectively). Therefore locality proceeds with individuals and datatypes as if they belonged to the interface signature anyway. Second, since LBMs focus on the logical axioms of an ontology, there is currently no support for specifying annotation properties in an interface signature. We do, however, believe that this would be a straightforward extension: remember that, after the logical module has been extracted, the relevant annotations are added. Among other things, this ensures that the above mentioned relevant versioning and incompatibility annotations are preserved in the module. If we additionally allow the specification of annotation properties in the interface signature, this will have the simple effect of an additional filter on the annotations to be included. Annotation properties relevant for versioning and incompatibility would then have to belong to the interface signature implicitly.

Directive versus integrity constraint. So far, we have treated the new imports statement as a directive: "Given the specified interface signature S and ontology $\mathcal{E}$, extract the S-module of $\mathcal{E}$ and import it into the current ontology $\mathcal{O}$." This is only one possible way of interpreting this statement. Another possibility is to additionally interpret it as an integrity constraint: "Given the specified interface signature S and ontology $\mathcal{E}$, make sure that all axioms in the current ontology $\mathcal{O}$ use no other terms from $\mathcal{E}$ than those in S." The idea behind this is that, if the right module extraction techniques are used, a module can only guarantee to cover all knowledge in $\mathcal{E}$ about the terms in the interface signature (and in the module itself if it is a self-contained module). Reusing other terms in $\mathcal{O}$ means that the knowledge in $\mathcal{E}$ cannot be guaranteed to be covered.

Unfortunately, the interpretation as an integrity constraint causes problems in unrestricted import scenarios. Suppose we have ontologies $\mathcal{O}_1, \ldots, \mathcal{O}_4$ with IRIs $I_1, \ldots, I_4$, a class $C \in \mathrm{sig}(\mathcal{O}_1)$, interface signatures S_1, S_1', S_2, S_3 with $C \notin S_1$, $C \in S_1'$, $C \notin S_2$, $C \in S_3$, and the following import statements:

(1)	`ImportModule`(S_1, I_1) in $\mathcal{O}_2$	(3)	`ImportModule`(S_2, I_2) in $\mathcal{O}_4$
(2)	`ImportModule`(S_1', I_1) in $\mathcal{O}_3$	(4)	`ImportModule`(S_3, I_3) in $\mathcal{O}_4$

See also the picture below, where Arrows point towards the importing ontology.



Then (2) would enable C to occur in $\mathcal{O}_3$, which would justify its containment in S_3. Therefore (4) would explicitly express the desire to have C in the module extracted from $\mathcal{O}_3$. On the other hand, since C is not in S_2, (3) read as an integrity constraint would forbid C to be reused in $\mathcal{O}_4$.

Now this problem could be resolved by giving permission a higher priority than prohibition, i.e., if C occurs in the signature of some import statement in $\mathcal{O}_4$ (explicit "permission" for its reuse), its absence in the other does not prohibit its use in $\mathcal{O}_4$ any longer. This might also work when more ontologies are involved and $\mathcal{O}_4$ contains more import statements. However, if we now deleted the "permitting" import statements, the use of C would suddenly be forbidden, but in order to be sure, we would have to trace its origin in the whole import structure.

We believe that this behaviour is highly confusing and undesired, and therefore advise against the interpretation as an integrity constraint. The only situation where it might work is "flat" import scenarios, e.g., simple collaborative development of an ontology where modules of an ontology are extracted to be maintained independently, and these modules are themselves not imported into any other ontology. Such a scenario has yet to be thoroughly described, which is not in the scope of this paper.

However, interpretating an import statement as a directive has a pitfall, too. Suppose ontology $\mathcal{O}_1$ imports a module $\mathcal{M}_2$ from an ontology $\mathcal{O}_2$ about vehicles for $S_2 = \{\mathsf{Car}, \mathsf{Wheel}\}$, and suppose $\mathcal{O}_1$ contains an axiom $\mathsf{Tram} \sqsubseteq \mathsf{Bus}$, where Tram and Bus occur in $\mathcal{O}_2$. From the perspective of $\mathcal{O}_1$, it is not clear whether these two terms occur in $\mathcal{M}_2$, and hence whether our axiom changes the knowledge of $\mathcal{O}_2$ about the imported terms. This indeed depends on the module extraction algorithm used. As one possible workaround, all terms in $\mathcal{O}_1$ that are not in S_2 could be treated as disjoint from the terms in $\mathcal{O}_2$.

Variations for convenience. It is possible to make the interface signature an optional parameter to the `ImportModule` statement, i.e., the production **directly-ImportsDocuments** can be extended as follows.

```
directlyImportsDocuments := {
    'Import' '(' IRI ')' |
    'ImportModule' '(' SignatureTerm {SignatureTerm} IRI ')' |
    'ImportModule' '(' IRI ')'  }
```

In case of the third line, the module is extracted for the interface signature comprising all entities from the external ontology that are reused in the current ontology. Again, the above guidelines regarding cyclic imports and import chains need to be taken into account.

Where is the module computed? Whenever an ontology is imported by signature, the corresponding module needs to be computed. There are several possible scenarios for this, and we consider it out of the scope of this paper to make an ultimate choice between them.

One scenario is to have the module computed by the (host for the) importing ontology. While this requires the download of all ontologies in the import closure of the importing ontology and a recursive module extraction, this is still more economic than simply importing the *whole* ontologies: first, they do not need to be kept in memory at once; and second, the import of the *modules* will increase the importing ontology only by an amount that is necessary.

Another scenario is to have the modules computed by the (hosts for the) imported ontologies. This distributed approach ensures that, in addition to the size of the resulting importing ontology, communication between the ontologies is reduced. For this worthwile alternative, protocols need to be devised.

Independently of these two scenarios, one could ask what happens if the imported ontology changes. Certainly all of its modules need to be recomputed. But this is not a problem, given the efficient algorithms available for extracting locality-based modules. Particularly for big, heavily re-used ontologies, the computational overhead of extracting all modules that are imported in other ontologies is clearly outweighed by the advantage of being able to work with the import closure of those ontologies *at all* because they are now small enough to be loaded, maintained, and classified.

4 Future work

Currently, modular import is not supported in OWL 2. It is conceivable to add this support to the next version of OWL if there will be a successor. However, even without having modular support specified in OWL, using the extension we have proposed is harmless: it can easily be built into tools as an unofficial extension, and tools that do not support it can simply ignore it. This can best be achieved by encoding `ImportModule` statements as `Import` statements with an additional annotation that contains the interface signature. We hope to implement support for this extension in the upcoming version of the OWL API.

The obvious next step is to evaluate the use of this extension experimentally. Furthermore, there needs to be support for ontology developers in choosing the desired terms from ontologies from which they want to import modules.

Finally, once the new mechanism for modular import is in place, it will facilitate collaborative ontology development. It remains to describe a precise methodology and develop appropriate tools.

References

[1] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: Theory and practice. *J. Artif. Intell. Research*, 31:273–318, 2008.

[2] B. Cuenca Grau, B. Parsia, and E. Sirin. Combining OWL ontologies using $\mathcal{E}$-connections. *J. of Web Semantics*, 4(1):40–59, 2006.

[3] F. Ensan and W. Du. An interface-based ontology modularization framework for knowledge encapsulation. In *Proc. of ISWC-08*, volume 5318 of *LNCS*, pages 517–532. Springer, 2008.

[4] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? A case for conservative extensions in description logics. In *Proc. of KR-06*, pages 187–197, 2006.

[5] J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, J. Oberthaler, and B. Parsia. The National Cancer Institute's thésaurus and ontology. *J. Web Sem.*, 1(1):75–80, 2003.

[6] E. Jiménez-Ruiz, B. Cuenca Grau, U. Sattler, T. Schneider, and R. Berlanga Llavori. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proc. of ESWC-08*, volume 5021 of *LNCS*, pages 185–199, 2008.

[7] B. Konev, C. Lutz, D. Walther, and F. Wolter. Logical difference and module extraction with CEX and MEX. In *Proc. of DL 2008*, volume 353. CEUR (`http://ceur-ws.org/`), 2008.

[8] B. Konev, C. Lutz, D. Walther, and F. Wolter. Semantic modularity and module extraction in description logics. In *Proc. of ECAI-08*, pages 55–59, 2008.

[9] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev. Minimal module extraction from DL-Lite ontologies using QBF solvers. In *Proc. of IJCAI-09*, pages 836–841, 2009.

[10] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proc. of IJCAI-07*, pages 453–458, 2007.

[11] B. Motik, P. Patel-Schneider, and B. Parsia. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax.* W3C Cand. Rec., 11 June 2009. `http://www.w3.org/TR/2009/CR-owl2-syntax-20090611/`.

[12] U. Sattler, T. Schneider, and M. Zakharyaschev. Which kind of module should I extract? In *Proc. of DL 2009*, volume 477. CEUR (`http://ceur-ws.org/`), 2009.

[13] K.A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the Amer. Med. Informatics Ass.*, 2000. Fall Symposium Special Issue.

[14] H. Stuckenschmidt, C. Parent, and S. Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS.* Springer, 2009.

[15] M. Zakharyaschev. Personal communication.