Incrementally Updateable and Persistent Decomposition of OWL Ontologies

Pavel Klinov[1], Chiara Del Vescovo[2], Thomas Schneider[3]

[1] University of Arizona, AZ, USA
pklinov@email.arizona.edu
[2] University of Manchester, UK
delvescc@cs.man.ac.uk
[3] Universität Bremen, Germany
tschneider@informatik.uni-bremen.de

Abstract The paper focuses on practical aspects of decomposing OWL ontologies into logically coherent fragments, called atoms, and maintaining large ontologies in the decomposed form. While the recently proposed atomic decomposition exhibits some very attractive formal properties and promises important applications, e.g., module extraction, its practical applicability has been far from clear. Two particular concerns are the necessity to recompute the decomposition upon every change to the logical content of the ontology and the lack of methods for managing the decomposed structure in a persistent storage. This paper describes tools which addresses both issues: an algorithm for incremental updates to the atomic decomposition (with a preliminary empirical evaluation) and an approach to its persistence in a RDF store or an XML database.[4]

Keywords: OWL, modularity, atomic decomposition, incremental updates, persistence

1 Introduction

The success of semantic technologies, and OWL in particular, in health, bio, and chemical informatics has led to the growing number of large ontologies. Some notably big ontologies, such as SNOMED CT or FMA, have more than 100K axioms while repositories like the NCBO BioPortal manage dozens of ontologies with more than 10K axioms, e.g., GO, ChEBI, GALEN, etc. Their growing size is increasingly becoming a problem since most of the knowledge is irrelevant to *specific* applications interested in a particular subarea of, say, anatomy.

The problem is especially relevant for applications concerned with reasoning since it is desirable to consider all knowledge necessary to draw all (or, at least, as many as possible) important conclusions while ignoring all irrelevant axioms. One example is solutions for Semantic Web service discovery and invocation,

[4] The presented methods are implemented and the code (including the evaluation code) is available online under the MIT license at:
http://code.google.com/p/owl-modularity-experiments/

such as SSWAP[5] [6], which often do reasoning over only few specific domain terms to enable a transaction between two services on Web. It is undesirable to load the entire ontology into memory, or transfer it over the network. One research direction to tackle this issue is ontology modularity [2]. It provides theoretical foundations and tool support for extracting *modules*, small fragments of ontologies that contain all relevant knowledge for a particular scenario, e.g., reasoning over a specific signature. The existing algorithms, namely, syntactic locality-based module extraction (ME) are linear in the size of the ontology and can readily support ME services for ontology repositories.[6]

However, just being linear may not be sufficient when the goal is to do a transaction-time reasoning over a few terms from, say, GO, to decide if two Web services can interact. First, loading the ontology into memory for subsequent ME leads to substantial delays while pre-loading all ontologies into the main memory can be a considerable burden for large repositories, such as BioPortal. Second, conventional ME algorithms examine each axiom for relevance which is also problematic when the number of axioms is high. These issues ask for a new approach to managing OWL ontologies that is different from the traditional way of storing them as plain collections of axioms (or triples).

One such approach can be based on the recently developed notion of atomic decomposition (AD) of OWL ontologies [5]. AD is a partition of an ontology into logically indivisible sets of axioms, called *atoms*, which can be regarded as basic blocks for modules. Among other advantages, AD greatly facilitates module extraction which can be done by quickly identifying relevant atoms. A recent study has showed that the state-of-the-art biomedical ontologies are well-decomposable into atoms and that AD-based ME is considerably faster than conventional ME [3]. It is, therefore, justified to consider maintaining OWL ontologies as decomposed structures.

Unfortunately, there are issues too. First, computing AD is a PTIME but nonetheless a computationally intensive procedure. This is problematic for frequently updateable ontologies since previously there has been no technique for updating AD other than recomputing it. A related issue is that there is no persistence mechanism for AD which partly defeats its advantages (i.e. not requiring to load ontologies into memory). This is particularly inconvenient for large repositories. These difficulties make AD impractical in many scenarios.

This paper shows how both problems can be addressed. First, it presents a preliminary algorithm for incremental updates to AD, namely, incorporating new axioms into the existing atomic structure. It proves some key facts establishing soundness of the method and then shows its substantial performance benefits for large, yet simple ontologies like GO or ChEBI. Second, it describes an approach to persisting AD into an XML or an RDF database so that decomposed ontologies can be conveniently maintained and used using familar semantic technologies. The presented methods are used in the module extraction service for the SSWAP discovery server and APIs.

[5] Simple Semantic Web Architecture and Protocol: `http://sswap.info`
[6] The TONES repository has provided a RESTful ME service since 2009.

2 Modularity and Atomic Decomposition

Assuming the reader is familiar with OWL and Description Logics (DLs) [1] we briefly sketch some of the central notions around locality-based modularity [2] and Atomic Decomposition [5,3]. We use $\mathcal{O}$ for ontologies, i.e. finite sets of axioms, and $\mathcal{M}$ ($\subseteq \mathcal{O}$) for modules. Moreover, we use $\widetilde{\alpha}$ for the signature of an axiom α, i.e., the set of class, property, and individual names used in α.

Given a set of terms, or seed signature, Σ, a Σ-module $\mathcal{M}$ based on deductive conservative extensions is a subset of an ontology $\mathcal{O}$ such that, for all axioms α with terms only from Σ, we have that $\mathcal{M} \models \alpha$ iff $\mathcal{O} \models \alpha$, i.e., $\mathcal{O}$ and $\mathcal{M}$ have the same entailments over Σ. Unfortunately, checking if a set of axioms is a module in this sense is at least as hard as reasoning or even undecidable for expressive DLs [7,9]. However, if we do not require modules to be minimal in terms of size or inclusion, we can define "good sized" approximations that are efficiently computable, e.g. *locality-based modules* (LBMs). Given an ontology $\mathcal{O}$ and a signature Σ, a single LBM can be extracted in time polynomial in the size of $\mathcal{O}$, see [2] for details. LBMs still provide strong logical guarantees: they capture *all* relevant entailments about Σ, although not necessarily *only* those [8]. Informally, LBMs are based on the notion of locality, that is, *relevance*, of an axiom to a given signature (axioms appear in the module extracted for the empty signature are called *global*). Syntactic module extraction algorithms are implemented in the OWL API[7] and SSWAP.

There are two main notions of LBMs, namely $\bot$ and $\top$: roughly speaking, a $\top$-module for Σ gives a view from above because it contains all subclasses of class names in Σ, while a $\bot$-module for Σ gives a view from below since it contains all superclasses of class names in Σ. Given a module notion $x \in \{\top, \bot\}$, we denote by x-mod$(\Sigma, \mathcal{O})$ the x-module of $\mathcal{O}$ w.r.t. Σ. The number of modules of $\mathcal{O}$ can be exponential in the size of $\mathcal{O}$ [4]. However we can focus only on *genuine* modules, i.e. modules that are not the union of two "$\subseteq$"-uncomparable modules. Such modules define a base for all modules, and remarkably the family of genuine modules is linear in the size of $\mathcal{O}$ [5].

Some sets of axioms never split across two modules [5], revealing a strong logical interrelation. For $x \in \{\top, \bot\}$, we call *x-atom* a maximal set $\mathfrak{a}^x \subseteq \mathcal{O}$ such that, for each x-module $\mathcal{M}^x$, either $\mathfrak{a}^x \subseteq \mathcal{M}^x$, or $\mathfrak{a}^x \cap \mathcal{M}^x = \emptyset$. The family of x-atoms of $\mathcal{O}$ is denoted by $\mathcal{A}^x(\mathcal{O})$ and is called *x-Atomic Decomposition (x-AD)*. If x or $\mathcal{O}$ are clear from the context, or irrelevant, we drop them.

Since every atom is a set of axioms, and atoms are pairwise disjoint, the AD is a partition of the ontology, and its size is at most linear w.r.t. the size of the ontology. Moreover, there is a 1-1 correspondence between atoms and genuine modules: for each atom $\mathfrak{a}$ we denote by $\mathcal{M}_\mathfrak{a}$ the corresponding genuine module, that is also the smallest module containing $\mathfrak{a}$. Then we can define a second logical relation between atoms: an atom $\mathfrak{a}$ is *dependent* on a distinct atom $\mathfrak{b}$ (written $\mathfrak{a} \succ \mathfrak{b}$) if $\mathcal{M}_\mathfrak{b} \subset \mathcal{M}_\mathfrak{a}$. Note that this property then holds for all modules containing $\mathfrak{a}$. The dependence relation $\succ$ on AD is a partial order (i.e., transitive,

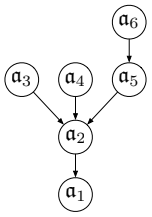
[7] `http://owlapi.sourceforge.net`

reflexive, and antisymmetric) and is computable in polynomial time [5]. Finally, the union of all atoms which a given atom $\mathfrak{a}$ depends on is called the *principal ideal* of $\mathfrak{a}$ (denoted as $(\mathfrak{a}]$). Interestingly, $\mathcal{M}_{\mathfrak{a}}$ and $(\mathfrak{a}]$ coincide [5].

Atoms can be seen as building blocks for modules: for each x-module $\mathcal{M}$ of an ontology $\mathcal{O}$, there are atoms $\mathfrak{a}_1, \ldots, \mathfrak{a}_\kappa$ in $\mathcal{A}^x(\mathcal{O})$ such that $\mathcal{M} = \bigcup_{i=1}^{n} \mathfrak{a}_i$. The converse does not hold, since not all combinations of atoms are modules. However, an algorithm to perform the extraction of a module from the AD of an ontology is described in [3].

Example 1. Consider the following ontology and its $\bot$-AD:

$\mathcal{O} = \{\alpha_1 = [\texttt{Animal} \sqsubseteq \exists \texttt{hasGender.Thing}],$
$\alpha_2 = [\texttt{Animal} \sqsubseteq \geq 1\,\texttt{hasHabitat.Thing}],$
$\alpha_3 = [\texttt{Person} \sqsubseteq \texttt{Animal}],$
$\alpha_4 = [\texttt{Vegan} \equiv \texttt{Person} \sqcap \forall \texttt{eats.}(\texttt{Vegetable} \sqcup \texttt{Mushroom})],$
$\alpha_5 = [\texttt{TeeTotaller} \equiv \texttt{Person} \sqcap \forall \texttt{drinks.NonAlcoholicThing}],$
$\alpha_6 = [\texttt{Student} \sqsubseteq \texttt{Person} \sqcap \exists \texttt{hasHabitat.University}],$
$\alpha_7 = [\texttt{GraduateStudent} \equiv \texttt{Student} \sqcap \exists \texttt{hasDegree.}(\{\text{BA, BS}\})]\}$



Here the $\bot$-atoms in the AD contain the following axioms respectively: $\mathfrak{a}_1 = \{\alpha_1, \alpha_2\}$, $\mathfrak{a}_2 = \{\alpha_3\}$, $\mathfrak{a}_3 = \{\alpha_4\}$, $\mathfrak{a}_4 = \{\alpha_5\}$, $\mathfrak{a}_5 = \{\alpha_6\}$, $\mathfrak{a}_6 = \{\alpha_7\}$.

3 Incremental Changes to Atomic Decomposition

This section describes the incremental AD algorithm and its theoretical basis. It also presents the results of the evaluation performed on fragments of GO and ChEBI ontologies which show that adding an axiom into an existing AD is orders of magnitude faster than recomputing it.

3.1 Theoretical Basis and the Algorithm

We first establish some facts which our algorithm for accommodating new axioms into an existing AD is based on. The key observation is that whenever new axioms are added to an ontology, atoms in the new AD are *conglomerates* of atoms in the previous AD, but existing atoms are never split into new atoms.[8] Intuitively, this follows from the monotonicity of locality-based modules [2] (in what follows we continue to consider the module notion $x \in \{\bot, \top\}$).

Lemma 2 (Atoms Do Not Split). *Let $\mathcal{O} \subseteq \mathcal{O}'$ be ontologies. For any atom $\mathfrak{a}$ in the AD of $\mathcal{O}$ and any atom $\mathfrak{b}'$ in the AD of $\mathcal{O}'$, either $\mathfrak{a} \subseteq \mathfrak{b}'$ or $\mathfrak{a} \cap \mathfrak{b}' = \emptyset$.*

Proof. Assume that there were atoms $\mathfrak{a}$ in the AD of $\mathcal{O}$ and $\mathfrak{b}'$ in the AD of $\mathcal{O}'$ such neither $\mathfrak{a} \subseteq \mathfrak{b}'$ nor $\mathfrak{a} \cap \mathfrak{b}' = \emptyset$. Then there are axioms $\beta \in \mathfrak{a} \cap \mathfrak{b}'$ and $\gamma \in \mathfrak{a} \setminus \mathfrak{b}'$. Let $\mathfrak{c}'$ be the atom in the AD of $\mathcal{O}'$ that contains γ. By proving that $\mathfrak{b}' = \mathfrak{c}'$, we will establish a contradiction. Hence, the assumption cannot hold.

Consider the genuine module x-mod$(\widetilde{\beta}, \mathcal{O}')$. Due to monotonicity of the module extraction process, this module contains x-mod$(\widetilde{\beta}, \mathcal{O})$, which in turn contains

[8] Yet another justification of the term "atom"!

$\mathfrak{a}$ [5]. Since $\gamma \in \mathfrak{a} \subseteq x\text{-mod}(\widetilde{\beta}, \mathcal{O})$, we get $\gamma \in x\text{-mod}(\widetilde{\beta}, \mathcal{O}')$. Therefore, the principal ideal of $\mathfrak{b}'$ contains $\mathfrak{c}'$. The same argument with β and γ (and $\mathfrak{b}'$ and $\mathfrak{c}'$) interchanged yields that the principal ideal of $\mathfrak{c}'$ contains $\mathfrak{b}'$. Hence, $\mathfrak{b}' = \mathfrak{c}'$. □

As a consequence of Lemma 2, every atom $\mathfrak{a}'$ of the extended ontology can never contain a *part* of any atom from the previous ontology. Hence, $\mathfrak{a}'$ is always a *conglomeration* of atoms from the previous ontology, possibly together with axioms that have been added.

Corollary 3. *Let $\mathcal{O}$, $\mathcal{O}_1$ and $\mathcal{O}' = \mathcal{O} \uplus \mathcal{O}_1$ be ontologies. Then every atom in the AD of $\mathcal{O}'$ is either*

- *the union of one or more atoms of $\mathcal{O}$, or*
- *the union of zero, one or more atoms of $\mathcal{O}$, enriched with axioms from $\mathcal{O}_1$.*

Unfortunately, the addition of α can also lead to conglomerates of several "old" atoms without α. The following example shows this behaviour and additionally illustrates that dependencies involving merged atoms in $\mathcal{O}'$ cannot be so easily obtained from the dependencies of the respective atoms in $\mathcal{O}$.

Example 4. Consider the ontology $\mathcal{O} = \{S \sqcap X \sqsubseteq Y,\ T \sqcap X \sqsubseteq Z\}$. In the $\bot$-AD of $\mathcal{O}$, the two axioms constitute two independent atoms $\mathfrak{a}, \mathfrak{b}$. However, if we add the axiom $\alpha = [S \equiv T]$, we obtain that the $\bot$-AD of $\mathcal{O}' = \mathcal{O} \cup \{\alpha\}$ contains two atoms: $\mathfrak{c} = \{\alpha\}$, and $\mathfrak{d} = \mathfrak{a} \cup \mathfrak{b}$. Moreover, $\mathfrak{d} \succ \mathfrak{c}$.

Next, we briefly describe Algorithm 1 for adding an axiom into an existing AD. Informally, its idea is to select a fragment of the AD which is *affected* by the addition (see Def. 5), re-decompose that fragment, and incorporate it back into the AD graph. Some more informal insights into various aspects of the algorithm are given below.

Algorithm 1 Main routine

Input: the ontology $\mathcal{O}$, its x-AD $\mathcal{A}(\mathcal{O})$, $x \in \{\top, \bot\}$, the new axiom α
Output: the updated x-AD $\mathcal{A}(\mathcal{O} \cup \{\alpha\})$
$\mathcal{O}' \leftarrow \mathcal{O} \cup \{\alpha\}$
$\mathsf{GA}(\mathcal{O})$ $\leftarrow$ the set of x-global axioms in $\mathcal{O}$
$\mathsf{GA}(\mathcal{O}')$ $\leftarrow$ the set of x-global axioms in $\mathcal{O}'$
if $\mathsf{GA}(\mathcal{O}') \neq \mathsf{GA}(\mathcal{O})$ **then**
    $\mathcal{A}(\mathcal{O}')$ $\leftarrow$ decompose($\mathcal{O}'$)
else
    $\mathsf{AFF}(\mathcal{O}, \alpha)$ $\leftarrow$ find-affected-atoms($\mathcal{A}(\mathcal{O})$, α, $\mathsf{GA}(\mathcal{O}')$)
    remove-atoms-from-AD($\mathsf{AFF}(\mathcal{O}, \alpha)$, $\mathcal{A}(\mathcal{O})$)
    $\mathcal{A}(\mathcal{O}^{AFF})$ $\leftarrow$ decompose($\alpha \cup \bigcup_{\mathfrak{a} \in \mathsf{AFF}(\mathcal{O}, \alpha)} \mathfrak{a} \cup \mathsf{GA}(\mathcal{O}')$)
    $\mathcal{A}(\mathcal{O}')$ $\leftarrow$ merge-new-AD-into-old($\mathcal{A}(\mathcal{O})$, $\mathcal{A}(\mathcal{O}^{AFF})$)
end if
return $\mathcal{A}(\mathcal{O}')$

Definition 5 (Affected Atoms). *Given an ontology $\mathcal{O}$, its x-AD $\mathcal{A}(\mathcal{O})$, and a new axiom α, the* ***affected atoms****, denoted as* $\mathsf{AFF}(\mathcal{O}, \alpha)$ *are those atoms whose corresponding genuine modules changed as a result of adding α, i.e.: if* $x\text{-mod}(\widetilde{\mathfrak{a}}, \mathcal{O} \cup \alpha) \supset x\text{-mod}(\widetilde{\mathfrak{a}}, \mathcal{O})$*, then* $\mathfrak{a} \in \mathsf{AFF}(\mathcal{O}, \alpha)$.

Algorithm 2 is the procedure for computing $\mathsf{AFF}(\mathcal{O}, \alpha)$. Due to Cor. 3 this could be done by selecting atoms whose corresponding genuine modules extended (but never split) as a result of the addition. The algorithm computes $\mathsf{AFF}(\mathcal{O}, \alpha)$ by first selecting atoms whose signature contains at least one minimal globalizing subsignature (MGS) of α (i.e., α is non-local w.r.t. their signature) and then taking all atoms in the same connected component of the AD graph. Obviously, this makes the set $\mathsf{AFF}(\mathcal{O}, \alpha)$ larger than it needs to be and, thus, the algorithm may re-decompose more axioms than needed. In future work we plan to rectify this. Note, however, that ADs for many state-of-the-art ontologies are very disconnected graphs (see [3]) for which this algorithm works very well, as shown in the next subsection.

Algorithm 2 find-affected-atoms

1: **Input:** the current x-AD $\mathcal{A}(\mathcal{O})$, the new axiom α, the set of global axioms GA
2: **Output:** the set of atoms $\mathsf{AFF} = \{\mathfrak{a} \mid \alpha \in x\text{-mod}(\widetilde{\mathfrak{a}}, \mathcal{O}')\}$
3: $\mathsf{AFF} \leftarrow \{\mathfrak{a} \in \mathcal{A}(\mathcal{O}) \mid \exists \Sigma \in \mathsf{MGS}(\alpha) \text{ s.t. } \Sigma \subseteq \widetilde{[\mathfrak{a}]} \cup \widetilde{\mathsf{GA}}\}$
4: **return** $\{\mathfrak{b} \mid \exists \mathfrak{a} \in \mathsf{AFF}$ and there exists an undirected path from $\mathfrak{a}$ to $\mathfrak{b}\}$

The procedures `remove-atoms-from-AD` and `decompose` are self-explanatory: the former removes the given atoms from the AD (including the dependencies), the latter invokes the standard (that is, non-incremental) AD algorithm for the given set of axioms. Algorithm 3 describes the important procedure `merge-new-AD-into-old` which merges the re-decomposed affected part into the existing AD (from which the affected part had been removed beforehand).

Algorithm 3 merge-new-AD-into-old

1: **Input:** the current x-AD $\mathcal{A}(\mathcal{O})$, the x-AD for the affected part $\mathcal{A}(\mathcal{O}^{AFF})$
2: **Output:** the updated x-AD $\mathcal{A}(\mathcal{O}')$
3: $\mathsf{NA} \leftarrow \bigcup_{\mathfrak{a} \in \mathcal{A}(\mathcal{O}^{AFF})} \mathfrak{a}$
4: $\mathcal{A}(\mathcal{O}') \leftarrow \mathcal{A}(\mathcal{O}) \cup \mathcal{A}(\mathcal{O}^{AFF})$
5: **for** each $\mathfrak{a} \in \mathcal{A}(\mathcal{O}^{AFF})$ **do**
6: **for** $\beta \in x\text{-mod}(\widetilde{\mathfrak{a}}, \mathcal{O}') \setminus \mathsf{NA}$ **do**
7: $\mathfrak{b} \leftarrow$ get-atom-for-axiom$(\beta, \mathcal{A}(\mathcal{O}'))$
8: set $\mathfrak{a} \succ \mathfrak{b}$ in $\mathcal{A}(\mathcal{O}')$
9: **end for**
10: **end for**
11: $\mathcal{A}(\mathcal{O}') \leftarrow$ transitive-reduction$(\mathcal{A}(\mathcal{O}'))$
12: **return** $\mathcal{A}(\mathcal{O}')$

Once the set of atoms for $\mathcal{A}(\mathcal{O}')$ has been determined, the rest is to figure out any new dependencies between atoms. Those could be of two sorts: dependencies between atoms in the affected part and dependencies between two atoms where

one is in the affected part and the other is not. The former have already been computed on the re-decompose step. The procedure `merge-new-AD-into-old` takes care of the latter. It relies on the following fact:

Lemma 6 (New Dependent Atoms Are Affected). *Let us consider* $\mathcal{O}' = \mathcal{O} \cup \{\alpha\}$*, and let* $\beta, \gamma \in \mathcal{O}$ *be two axioms such that:*

1. β *and* γ *belong to two independent atoms* $\mathfrak{b}$ *and* $\mathfrak{c}$ *in* $\mathcal{O}$
2. β *and* γ *either belong to the same atom* $\mathfrak{b}'$ *in* $\mathcal{O}'$*, or they belong to two distinct atoms* $\mathfrak{b}'$ *and* $\mathfrak{c}'$ *such that* $\mathfrak{b}' \succ \mathfrak{c}'$.

Then, $\mathfrak{b} \in \mathsf{AFF}(\mathcal{O}, \alpha)$.

Proof. By condition 2. we know that $x\text{-mod}(\widetilde{\mathfrak{b}}, \mathcal{O}') \ni \gamma$, whilst by condition 2. $x\text{-mod}(\widetilde{\mathfrak{b}}, \mathcal{O}) \not\ni \gamma$. By definition, the atom $\mathfrak{b}$ is affected iff $x\text{-mod}(\widetilde{\mathfrak{b}}, \mathcal{O}') \neq x\text{-mod}(\widetilde{\mathfrak{b}}, \mathcal{O})$. Hence, $\mathfrak{b} \in \mathsf{AFF}(\mathcal{O}, \alpha)$. □

The lemma states that an unaffected atom may not become dependent as a result of the addition since in that case it must be affected according to Def. 5. The merge procedure goes through the set of affected atoms and checkes whether their genuine modules should contain any axioms from the unaffected part. If it is the case, then a new dependency is added.

Termination of Algorithm 1 is fairly obvious: all its steps are entirely syntactic procedures which iterate over finite sets (e.g., affected atoms, axioms in a module, etc.). The only aspect which requires a comment is the search through the set of minimal globalizing subsignatures of the new axiom (Algorithm 2, line 3). In principle, this set may not be polynomial in the number of terms in the axiom. However, it is such in most practical cases and, if not, this step can be replaced by a PTIME check for syntactic locality of the axiom w.r.t. the atom signature union the signature of the global axioms. It is easy to see that all other procedures terminate in polynomial time.

3.2 Evaluation

In this section we present the results of an experiment which compares performance of adding a new axiom into an existing $\bot$-AD to performance of re-decomposing the ontology upon adding a new axiom. Our aim is demonstrate the utility of the approach on typical state-of-the-art biomedical ontologies, which are mainly large but simple concept hierarchies without global axioms. For that purpose we select random subsets of such important ontologies as GO and ChEBI (note that GO also serves as a perfect representative of the whole family of similar ontologies, including the Plant Ontology, the Sequence Ontology, etc.).

We use the following evaluation methodology for both GO and ChEBI: First, we generate 30 random fragments of the ontology, 10 for each size of 1K, 5K, and 10K axioms. Second, AD is computed for each. Third, for each fragment we randomly select 10 axioms which will be added to the AD. Importantly, we select only those axioms whose signature contains terms *already* occurring in the fragment. It makes much more likely that the additions do not simply lead

to new atoms without any logical connections to the existing atoms. For every addition we compare the average time to incorporate the axiom into the AD and the average time to re-decompose the fragment with the new axiom. We also record the size of the affected part (the total number of axiom in affected atom over the size of the fragment). The results are presented in the following table.

Size	GO			ChEBI		
	Avg. decomp. time (ms)	Avg. add time (ms)	Avg. % of affected atoms	Avg. decomp. time (ms)	Avg. add time (ms)	Avg. % of affected atoms
1000	118	17	0.55	141	21	0.11
5000	1402	38	0.21	1562	45	0.58
10000	6757	93	0.19	8482	1025	2.93

As expected, the results show a huge benefit from using the incremental algorithm. Most of the time less than 1% of existing atoms are affected by adding a new axiom into the existing fragment which leads to quick addition of the axiom into the AD. One may argue that such ontologies as GO and ChEBI represent the "happy path" for the current incremental algorithm since their AD graphs are largely disconnected whereas the algorithm would behave substantially worse on ontologies with complex dependencies between atoms. While true, we believe that the speed-up for simple, yet common ontologies is a sufficient reason for adopting the algorithm. We leave a more thorough evaluation, including investigation of worst cases, to a future work after we have addressed the above mentioned issues, in particular, the need to re-decompose the entire connected component for each affected atom.

4 Persistent Atomic Decomposition

Another major reason why AD may be considered impractical is the lack of engineering solutions for persisting the decomposition and using it without loading all atoms into memory. Let us go back to our basic scenario of an ontology repository with a ME service. Each ontology could be decomposed such that the repository may serve ME requests extremely fast by selecting relevant atoms for the signature. But maintaining the decomposition of every repository ontology in the main memory can be a scalability and robustness concern, especially for repositories with hundreds of ontologies (the BioPortal scale).[9] What is needed is a solution for keeping all data structures in the AD in a scalable yet performant storage, e.g., a database, which would enable fast execution of the common operations. The operations include the following:

- Finding directly relevant atoms for a given signature Σ: $\{\mathfrak{a} \mid \exists \alpha \in \mathfrak{a}$ s.t. α is non-local w.r.t $\Sigma\}$. This is the key operation during the AD-based ME algorithm [3].

[9] In fact, this issue also arises when ontologies are maintained in a traditional way as soon as loading/updating axioms in text (or XML) files becomes a bottleneck.

- Finding all atoms that a given atom depends (or is dependent) on.
- Loading all axioms for a given atom.

These operations mandate the data structures which need to be maintained. In particular, the first operation requires i) labels which store minimal set signatures MSS for each atom and ii) an extra term index which maps every term onto a set of atom whose MSS-labels include that it (intuitively, the set of atoms which are relevant to the term). The second and third operations suggest management of persistent graph structures in the form of adjacency lists and a certain grouping of axioms belonging to the same atom.

For persistence purposes it is handy to regard AD as a tuple $\langle \mathcal{G}, \mathcal{L}, \mathsf{TI}, \mathsf{GI} \rangle$, where $\mathcal{G} = (\mathcal{A}, \succ)$ is the AD graph, $\mathcal{L}$ is the labeling function (e.g., in case of MSS labels it maps each atom to its MSS), TI is a term index which depends on $\mathcal{L}$ (see previous paragraph in case $\mathcal{L}$ is MSS labeling), and GI is global information which pertains to the AD as a whole (module type, the set of global axioms, the set of non-logical axioms and syntactic tautologies, etc.). Next, we describe two particular implementations of managing this information in a persistent storage using conventional technologies: XML and RDF databases.

4.1 Atomic Decomposition over XML DB

XML databases are databases which store XML natively, i.e. do not map XML documents into any sort of other structure, e.g., relations. Their popularity has been growing for the last decade which, in particular, resulted in a certain unification in the form of commonly supported APIs, such as the XML:DB API. Most XML DBs provide support for XML collections (which can contain XML documents or other collections), XPath expressions, and XQuery queries for accessing collections, documents, and data inside the documents.

The AD data structures map naturally onto hierarchical XML collections. Our implementation does so in the following way (let `ont-log-URI` be the logical URI of the ontology):

- `/ont-log-URI`: root XML collection for the ontology;
- `/ont-log-URI/global.xml`: XML document which stores GI;
- `/ont-log-URI/atoms`: collection of all XML documents representing atoms. Each atom is a separate document, e.g., `/ont-log-URI/atoms/atom_i.xml`, in one of the XML-based syntaxes of OWL (RDF/XML or OWL/XML);
- `/ont-log-URI/graph.xml`: XML document defining the graph structure of the AD (that is, $\mathcal{G}$);
- `/ont-log-URI/labels.xml`: XML document for storing labels.

The graph document can be easily maintained in one of the standard XML-based graph languages (in our case, GraphML). The graph structure may always be loaded into memory or be maintained in the DB and loaded in a lazy fashion. The term index TI may be stored in a separate document or computed on-the-fly based on labels. Format of the labels document depends on $\mathcal{L}$. Listing 1 is an

Listing 1 A fragment of labels.xml representing two MSS labels for an atom

```
<labels>
  <atom id="1">
    <label>
      <signature>
        <term>http://purl.org/obo/owl/GO#GO_0005262</term>
        <term>http://purl.org/obo/owl/GO#GO_0005261</term>
        ...
      </signature>
      <signature>
        <term>http://purl.org/obo/owl/GO#GO_0006641</term>
        <term>http://purl.org/obo/owl/GO#GO_0006639</term>
        ...
      </signature>
    </label>
  </atom>
  ...
</labels>
```

example for MSS-based labels (for each labeling function $\mathcal{L}$ one can create an XML Schema document to make the format explicit).

Once the (MSS-labeled) decomposition is stored in such XML collections, it is very easy to use, for example, for module extraction. The AD-based ME algorithm (see [3]) is based on labels and does not require explicit axiom locality checks. As such, it boils down to using the term index and labels to find directly relevant atoms for a signature, the graph structure to find dependent atoms, and finally, loading axioms for the found atoms. All these operations can be implemented simply by using XPath/XQuery capabilities of modern XML DBs.

4.2 Atomic Decomposition over RDF DB

It is often the case that ontology providers prefer RDF to bare XML for the underlying storage format for OWL ontologies,[10] so we show how that can be done too. The main difference is that instead of collections and XML documents the AD data structures are mapped onto named graphs and triples.

One particular inconvenience with using RDF is its lack of support for hierarchical data organization. While named graphs are great for separating atoms from each other they are not, unlike XML collections, meant to be nested so grouping all atoms for the same AD is not easy.[11] Therefore, we dedicate each AD its own RDF DB. In our ontology repository scenario this even helps scalab-

[10] This appears to have roots in the famous Semantic Web cake according to which RDF is the basis of OWL. We believe that translation of OWL axioms into RDF triples is often unnecessary and OWL/XML (which is natural to store in an XML DB) is often a more sutable syntax than RDF-based syntaxes. However, these issues are beyond the scope of this paper.

[11] One can still achieve this by using graph URIs sharing a common prefix.

ility w.r.t. the number of ontologies at the cost of maintaining a special (system) DB which maps each ontology URI to the DB which stores the decomposition.

The AD data structures are represented as follows. Each atom is assigned a unique URI, which is the URI of the named graph that stores all axioms for the atom. The set of global axioms is also stored in a separate named graph. Axioms are translated into RDF triples. Then we introduce a special RDF vocabulary for representing the AD graph structure, global information, and labels. All data except of axioms can be stored in the default graph since ADs for different ontologies never share a DB. The vocabulary is a tiny RDFS ontology which provides classes (`Atom`, `Label`, `MSSLabel`, etc.) and object properties (`dependsOn`, `hasLabel`, `hasTerm`, etc.) for representing and verifying consistency of stored ADs. Example is shown in Listing 2.

Listing 2 A fragment of RDF representation of an atomic decomposition

```
ad:atom_1 rdf:type ad:Atom
ad:atom_2 rdf:type ad:Atom
ad:atom_1 ad:dependsOn ad:atom_2
ad:atom_1 ad:hasLabel ad:label_1
ad:label_1 rdf:type ad:MSSLabel
ad:label_1 ad:hasSignature ad:mss_sig_1
ad:mss_sig_1 ad:hasTerm GO:GO_0005262
...
<http://sswap.info/ad-in-rdf/atom_1>
    {
      GO:GO_0005262 rdfs:subClassOf GO:GO_0005261 .
      ...
    }
```

Triples in Listing 2 say, respectively, that `atom_1` depends on `atom_2`, `atom_1` has a MSS label with a signature (`mss_sig_1`) which contains a GO term `GO_0005262`, and `atom_1` stores the axiom that `GO_0005262` $\sqsubseteq$ `GO_0005261`.

The RDF representation also supports the same basic operations, e.g., finding all atoms relevant to a term, just instead of XQuery it is done via SPARQL. Again, the structure of the DB is less transparent than for XML DB but this is addressable via a suitable API. On the other hand, it is fairly possible that modern RDF stores outperform XML DBs on large datasets although we have not done such an evaluation.

5 Summary

The solutions presented in this paper—incremental updates and persistence for AD—aim to make ontology decomposition a *practically* feasible and attractive technique. AD is a very promising tool for module extraction, ontology comprehension, and collaborative development, but it has to efficiently support all traditional use cases, most importantly editing and browsing, to be widely adopted. Ideally, the physical organization of an ontology (whether it is decomposed or not, stored in a DB or in a text file) should be separated from its logical

representation, i.e., a set of axioms. The presented methods are the first step towards that goal.

Importantly, the incremental updates algorithm and persistent management play well together. Persistent ADs, whether XML or RDF based, efficiently support all the operations needed for the incremental algorithm, that is, finding (affected) atoms, inserting axioms into atoms, and adding new atoms and dependencies into the graph. A common interface may completely hide the persistence aspects from the updating algorithm. In fact, incremental updates to a persistent AD are even more important than to a non-persistent one since it reduces the amount of IO in addition to avoiding full re-decomposition.

Since this is, indeed, the first step, there is a lot of room for improvement, especial regarding incremental updates. This paper shows that it works well on large and simple ontologies but there are problems with ontologies whose AD graphs are highly connected. The algorithm is naïve in the sense that it simply re-decomposes the affected part of the AD. A better way would be to avoid the re-decomposition step with a more clever procedure for distinguishing between possible consequences of adding an axiom (creating a new atom, adding the axiom into an existing atom, or merging atoms). Finally, we plan to investigate updates to AD upon deletion of an axiom.

Acknowledgements This material is based upon work supported by the National Science Foundation (NSF) under grant #0943879 (PI: Damian Gessler) and the NSF Plant Cyberinfrastructure Program (#EF-0735191).

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. of Artif. Intell. Research 31, 273–318 (2008)
3. Del Vescovo, C., Gessler, D., Klinov, P., Parsia, B., Sattler, U., Schneider, T., Winget, A.: Decomposition and Modular Structure of BioPortal Ontologies. In: Proc. ISWC-11 (2011)
4. Del Vescovo, C., Parsia, B., Sattler, U., Schneider, T.: The modular structure of an ontology: an empirical study. In: Proc. of DL 2010. `ceur-ws.org` (2010)
5. Del Vescovo, C., Parsia, B., Sattler, U., Schneider, T.: The modular structure of an ontology: Atomic decomposition. In: Proc. of IJCAI-11. pp. 2232–2237 (2011)
6. Gessler, D., Schiltz, G., May, G., Avraham, S., Town, C., Grant, D., Nelson, R.: SSWAP: A simple semantic web architecture and protocol for semantic web services. BMC Bioinformatics 10, 309 (2009)
7. Ghilardi, S., Lutz, C., Wolter, F.: Did I damage my ontology? A case for conservative extensions in description logics. In: Proc. of KR-06. pp. 187–197 (2006)
8. Jiménez-Ruiz, E., Cuenca Grau, B., Sattler, U., Schneider, T., Berlanga Llavori, R.: Safe and economic re-use of ontologies: A logic-based methodology and tool support. In: Proc. of ESWC-08. LNCS, vol. 5021, pp. 185–199 (2008)
9. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: Veloso, M.M. (ed.) Proc. of IJCAI-07. pp. 453–458 (2007)