Query-Based Access Control for Ontologies

Martin Knechtel[1] * and Heiner Stuckenschmidt[2]

[1] SAP Research Center Dresden, Germany
martin.knechtel@sap.com
[2] Computer Science Institute, University of Mannheim, Germany
heiner@informatik.uni-mannheim.de

Abstract. Role-based access control is a standard mechanism in databases and information systems. Based on the role a user has, certain information is kept from the user even if requested. In the presence of ontology-based information, deciding which information can be given to a user without revealing secrets is more difficult as the user might be able to infer secret information from query results using logical reasoning. In this paper, we present two approaches to solving this problem: query rewriting and axiom filtering and show that while both approaches prevent the unveiling of secret information, axiom filtering is more complete in the sense that it does not suppress information the user is allowed to see while this happens frequently in query rewriting. Axiom filtering requires that each axiom carries a label representing its access level. We present methods to find an optimal axiom labeling to implement a query-based policy and report experiments on real world data showing that a significant number of results are retained using the axiom filtering method.

1 Motivation

Access control is an essential operation in standard information systems to prevent unauthorized access and use of information from the system. Access control basically consists of two tasks: the first is to determine the status of the person who is asking for access to information and once this has been done to determine - based on the status - what information should be given to this person. As semantic technology becomes used in real applications, access control also becomes an issue for ontology-based information. In a traditional information system, where all the available information is stored explicitly, it is possible to simply label information items with information about roles, users must have, to be allowed to receive this particular information. With semantic information, this approach does not work anymore, as new information can be derived from existing statements using the definitions in the ontology leading to a problem known as the *'inference problem'* [5]: avoiding a situation where a user can infer information he should not have access to using information he is allowed to access. To make the problem well defined, we assume that the user has the same ability to derive information as the information system.

* This research was partly funded by the German Federal Ministry of Economics and Technology under the promotional reference 01MQ07012 and the German Federal Ministry of Education and Research under grant number 01IA08001A.

In this paper, we compare two existing proposals for solving the inference problem. The first approach proposed in [4] is based on the idea of rewriting user queries based on the role a user has in such a way that the result to the rewritten query only returns information the user is allowed to see. This approach assumes that access restrictions are defined in terms of concepts and relations certain users are allowed to see. The other approach proposed in [1] uses an a priori labeling of axioms in the ontology to consistently derive labels for implied information as well. Thus it determines access rights on the level of axioms in the ontology. Our comparison of the two approaches concludes that axiom filtering is more general since it is independent of the concrete ontology language which makes the approach preferable in many situations. Moreover it provides a higher availability in the sense that more information can be made publicly available which cannot be used to infer any secret information. However it requires an a priori labeling of axioms, which is impractical to be created manually. Obviously, it is much more natural to formulate access restrictions in terms of a query. We provide algorithms to compute a changed axiom labeling based on a query-generated set of access restriction goals. Computing such a change set has been solved in [9, 10] for single goals. Based on this we compute a change set for multiple goals simultaneously. We provide conflict resolution strategies for cases where the goal set contains conflicts. Our empirical results show that our algorithms behave well with practical ontologies, axiom filtering provides higher information availability compared to query rewriting and conflicts can be resolved by our resolution strategies.

2 Preliminaries

2.1 Ontologies

Ontologies are formal descriptions of the terminology used in an application domain. A number of logical languages have been proposed for representing ontologies. In this paper, we only consider sublanguages of the Web Ontology Language OWL that can be translated to Description Logics (DL).

Formally, an *ontology* O is a finite set, whose elements are called *axioms*, such that every subset of an ontology is itself an ontology. If $O' \subseteq O$ and O is an ontology, then O' is called a *sub-ontology* of O. One can distinguish ABox axioms A and TBox axioms T and let $O = T \cup A$. An ontology language specifies which sets of axioms are admitted as ontologies. For instance, given a Description Logic $\mathcal{L}$ (e.g., the DL $\mathcal{SHOIN}(\mathbf{D})$ underlying OWL DL), an ontology is a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$, concept assertion axioms of the form $C(a)$ and role assertion axioms of the form $R(a, b)$ for $\mathcal{L}$-concept descriptions C, D, role R and individuals a, b. In order not to mix user roles and DL roles, we stick to the OWL lingo and call DL roles from now on *properties*. The signature $sig(O)$ of an ontology is the set of all concept and role names occurring in its axioms. Given an ontology language, a *monotone consequence relation* $\models$ is a binary relation between ontologies O and *consequences* c such that if $O \models c$, then for every ontology $O' \supseteq O$ it holds that $O' \models c$. If $O \models c$, we say that *c follows from* O or that O *entails* c. Often, a consequence c already follows from a subset $S \subseteq O$ of the axioms in the ontology. We call such a subset an explanation for $O \models c$ if there is no subset $S' \subset S$ such that $S' \models c$. Note that for one consequence there might be multiple explanations.

A query to an Ontology is a conjunction $Q = A_1, \cdots, A_n$ of OWL axioms over sig(O) and variables not necessarily from O. For a concrete definition of the form of axioms see [12]. The set of variables occurring in Q is denoted as $var(Q)$. Let $ind(O)$ be the set of individuals in O, then the result of a query is the set of all mappings $\mu : var(Q) \rightarrow ind(O)$ assigning individuals from O to variables in Q. An answer $\mu(Q)$ to a query Q is an instantiation of all variables in the query, so that $O \models \mu(Q)$ [12]. Note that there might be several possible μ for one query.

2.2 Access Control

Access control systems enable the regulation of access to protected resources (i.e. objects) in distributed systems by subjects such as users or system processes. They can be categorized in discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC) models. In DAC-based systems, the permissions to access an object are defined by its owner. In MAC models, the system determines the access to objects either by utilizing access rules or lattices for assigning permissions to subjects. It thus removes the ability of the users to control access to their resources. RBAC systems finally remove the explicit use of subjects within access rules or lattices and replace them with roles, which form a logical group of a number of subjects. In fact, permissions are assigned to roles and the subjects are assigned members of a number of roles. Thus changes of single subjects do not necessarily have consequences in the actual access control policies. On the most fine-grained level, permissions can be defined on the level of axioms, or on the level of query responses.

2.3 Access Restrictions as Queries

Assume we want customers and employees to query information from a product ontology. From Example 1, employees have full access and we do not want customers to see if any product gets an increased price soon. This restriction could be defined by enumerating all query responses except the price increase as permissions and assigning them to the respective user role. There are two problems with this approach. First of all, the price increase can still be inferred if the axioms of O can be queried. Further, enumerating all query responses, however, is not really feasible in practice and ask for more efficient ways of specifying these restrictions.

Example 1. Let O be an ontology from a marketplace in the Semantic Web with the following axioms

$a_1 : \mathit{EUecoService} \sqcap \mathit{HighperformanceService}(\mathit{ecoCalculatorV1})$
$a_2 : \mathit{HighperformanceService}$
$\quad \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}$
$a_3 : \mathit{EUecoService} \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}$
$a_4 : \mathit{ServiceWithLowCustomerNr} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}$
$a_5 : \mathit{LowProfitService} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}$

The consequence c_1 : $\mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$ follows from each of the explanations $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$. The consequence c_2 : $\mathit{LowProfitService}(\mathit{ecoCalculatorV1})$ follows from each of the explanations $\{a_1, a_2\}, \{a_1, a_3\}$. Three more instance assertions of individual $\mathit{ecoCalculatorV1}$ to the concepts $\mathit{EUecoService}, \mathit{HighperformanceService}, \mathit{ServiceWithLowCustomerNr}$ are consequences of O.

A way is to define permissions intentionally in terms of queries over the signature of the ontology. More specifically, we can describe facts that should not be accessible by a certain role in terms of a set of axioms - the same kinds of axioms used in queries - whose instantiations should not be derivable from query results. In the case of the example above, we could formulate the following access restriction for customers:

$$\mathit{ServiceWithComingPriceIncrease}(x)$$

stating that for no instance x in the ontology it should be possible to infer $\mathit{ServiceWithComingPriceIncrease}(x)$. The axiom addresses a set of consequences of O, in order to avoid addressing them one by one.

3 Enforcing Access Restrictions

There are different ways for implementing access control for ontological knowledge. While query rewriting extends a user's query to include all access restrictions, axiom filtering only allows a subset of the ontology to be used to answer the unchanged query.

3.1 Access Control by Query Rewriting

One option for enforcing access restrictions is by means of query rewriting. This approach has been proposed in [4] as a suitable way for enforcing access restrictions in the context of SPARQL queries, while the TBox is assumed to be completely public. Similar approaches are also allowing to hide TBox parts [7], or to define not the restrictions but the permissions by a query [3]. The idea in [4] is to automatically add filter conditions to the query that suppress such answers the user is not supposed to see. Given a Query Q and a set of access restrictions $\{AR_1, \cdots, AR_n\}$ that apply to the current user, the query can be rewritten to a new query that is defined as:

$$Q \wedge \neg AR_1 \wedge \cdots \wedge \neg AR_n$$

Where the junction of two queries $Q_1 \wedge Q_2$ is the junction of all contained query axioms $\bigwedge_{q \in Q_1} q \wedge \bigwedge_{q \in Q_2} q$ [12]. This way of rewriting the query based on the access restrictions of the individual users effectively prevents the system from giving away restricted information. In particular, using query rewriting, the effective answer to a query is

$$\{\mu(Q) | O \models \mu(Q \wedge \neg AR_1 \wedge \cdots \wedge \neg AR_n)\}$$

It however comes with a problem: it hides more information than necessary. In particular, in the example above where we want to hide from customers that some product is increased in price, the query rewriting approach hides too much information. If a customer for instance asks the system for all high performance services, thus $Q = \mathit{HighperformanceService}(x)$, this query will be rewritten to $\mathit{HighperformanceService}(x) \wedge \neg \mathit{ServiceWithComingPriceIncrease}(x)$. This query will only return high performance services which will not be increased in price. This is unfortunate, because the information that $\mathit{ecoCalculatorV1}$ is a high performance service was not supposed to be hidden. Similarly querying for instances the remaining concepts in $sig(O)$ are filtered, i.e. none of the 5 queries will yield a result.

3.2 Access Control by Axiom Filtering

A framework to control access to an ontology's axioms is introduced in [1]. In contrast to the query rewriting approach above, the TBox is not assumed to be completely public. The idea is to label each axiom with a certain access restriction. Users are labeled with the restrictions they are allowed to see. The approach is to use a labeling lattice $(L, \leq)$; i. e. a set L of labels together with a partial order $\leq$ such that every finite set of labels has a join ($\oplus$, supremum, least upper bound) and a meet ($\otimes$, infimum, greatest lower bound) w.r.t. $\leq$. Every axiom a in the ontology O is assumed to have a label $\mathsf{lab}(a) \in L$, and each user receives also a label $\ell \in L$. The sub-ontology to which a user with label ℓ has access is defined as

$$O_{\geq \ell} := \{a \in O \mid \mathsf{lab}(a) \geq \ell\}.$$

The sub-ontologies $O_{\ngeq \ell}, O_{\nleq \ell}$ etc. can be defined analogously. Applied to our scenario with the user roles customer (ℓ_C) and employee (ℓ_E), let the labeling lattice be $(L, \leq)$ with $L = \{\ell_C, \ell_E\}$ and $\leq = \{(\ell_E, \ell_C)\}$. Let the labeling function lab assign label ℓ_C to axioms a_1, a_2, a_3 and ℓ_E to axioms a_4, a_5. Employees can see $O_{\geq \ell_E} = \{a_1, a_2, a_3, a_4, a_5\}$, i.e. the complete ontology. Customers can see $O_{\geq \ell_C} = \{a_1, a_2, a_3\}$. Intuitively, the access restriction to a consequence, called boundary, should be based on the access restriction of its implying axioms. The access restriction for a consequence with multiple explanations should be the least restrictive of all explanations and within one explanation the most restrictive of all axioms. Formally, a consequence c with n explanations $S_1, \ldots, S_n$ has boundary $\bigoplus_{i=1}^{n} \bigotimes_{a \in S_i} \mathsf{lab}(a)$. In our example, each of the four explanations for c_1 has label $(\ell_C \otimes \ell_C \otimes \ell_E) = \ell_E$, thus the boundary is ℓ_E, i.e. employees can see it but customers not. Consequence c_2 has boundary ℓ_C, i.e. employees and customers can see it. Apart from c_1, c_2, instance relationships to the three remaining concepts in $sig(O)$ have boundary ℓ_C as can be verified easily. A customer querying for instances of the five concepts in the ontology will get no answer for $Q = \mathit{ServiceWithComingPriceIncrease}(x)$ but will get an answer for the four remaining queries. So axiom filtering provides $4/5$ answers, while query rewriting provides $0/5$ answers.

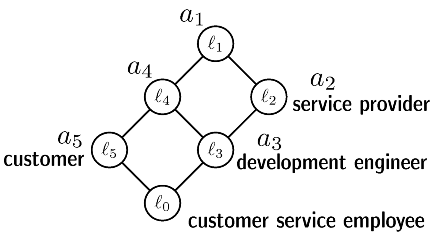

Fig. 1. Lattice $(L, \leq)$ with 4 user labels and an assignment of 5 axioms to labels

3.3 Discussion

As we have seen, the query rewriting approach is a way of ensuring that no classified information is given to users that do not have the permission to see it, but we have also seen that this approach is suboptimal with respect to availability in the sense of preserving maximal access to non-restricted information. We have further seen that axiom filtering allows higher availability of non-secret information while still ensuring that secret information is only given to intended users. However this requires an appropriate axiom labeling.

Previous work focused on labeled ontologies focused on computing a consequence's label based on axiom labels [1] and on repairing the axiom labeling so that a consequence's label changes to a specified one [10, 9]. However, access restrictions in the form of queries might require changing labels of multiple consequences simultaneously. Such a mechanism will be presented in the next section. Our main quality criterion for the algorithms is information availability. In the empirical evaluation we measure how many knowledge is additionally accessible with axiom filtering compared to query rewriting.

4 Optimal Axiom Labeling for implementing Access Control

In the last section we have only shown that there is an axiom labeling to enforce access restrictions for a selected example. Now we will elaborate how to compute it in general. We are starting from an arbitrary label assignment, and change it in a minimal way so that a given access restriction is enforced.

Example 2. We continue Example 1. Let $(L, \leq)$ be the lattice shown in Figure 1, where valid user labels are $\ell_0, \ell_2, \ell_3, \ell_5$ which represent user roles as illustrated. The condition for a valid user label is the join prime property discussed in [1]. Let O of Example 1 be a labeled ontology where the function lab assigns to each axiom a_i the label ℓ_i as shown in Figure 1. The computed boundary is ℓ_3 for c_1, since $= (\ell_1 \otimes \ell_2 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_2 \otimes \ell_5) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_4) \oplus (\ell_1 \otimes \ell_3 \otimes \ell_5)$. It is ℓ_2 for c_2, since $= (\ell_1 \otimes \ell_2) \oplus (\ell_1 \oplus \ell_3)$. For users ℓ_0 and ℓ_3, consequences c_1 and c_2 are visible. For user ℓ_2, only c_2 is visible.

We now define a notion for changing an axiom label assignment. Beforehand, we define the function lbl in order to address computed boundaries of consequences in a convenient way.

Definition 1 (Consequence Labeling Function). *Let O be a labeled ontology, $(L, \leq)$ a labeling lattice, $\mathsf{lab} : O \to L$ a labeling function. The* consequence labeling function *$\mathsf{lbl} : \{c \mid O \models c\} \to L$ assigns labels to consequences and is defined as $\mathsf{lbl}(c) =$ computed boundary of c.*

Definition 2 (MCS). *Let O be an ontology, c any consequence of O, $(L, \leq)$ a lattice, lab a labeling function, G a set of goals of the form (c, ℓ_g) with goal label ℓ_g for consequence c, M a set of assignments (a, ℓ) of label ℓ to axiom a. The* modified assignment *lab_M is defined to be*

$$\mathsf{lab}_M(a) = \begin{cases} \ell, & \textit{if } (a, \ell) \in M, \\ \mathsf{lab}(a), & \textit{otherwise.} \end{cases}$$

The respective consequence labeling function lbl_M is given by Definition 1. The set M is called multiple change set *(MCS) iff for any $c, (c, \ell_g) \in G : \mathsf{lbl}_M(c) = \ell_g$ and there is no $M' \subset M$ with $\mathsf{lbl}_{M'}(c) = \ell_g$.*

Whether we can find a lab_M fulfilling a given goal set is independent of the label assignment lab we start from. For default deny-all behavior, we start with all axioms assigned to the bottom lattice element. For default allow-all behavior, we start with all axioms assigned to the top lattice element. We will now introduce the computation of a change set for one goal and building on that introduce the computation of a MCS.

4.1 Computing a Change Set for one Goal Label

If G is the singleton set of only one tuple (c, ℓ), computing a multiple change set boils down to computing a change set (CS) which has been introduced in our prior work in [10, 9]. For every CS $S \subseteq O$ there is a MCS $M := \{(a, \ell_g) \mid a \in S\}$ and $\mathsf{lbl}_M(c) = \ell_g$ holds. The computation of a CS exploited main ideas from axiom-pinpointing [8, 2] and we presented a black-box approach that yields the desired set. Intuitively, a consequence c needs to be made more public if $\ell_g > \mathsf{lbl}(c)$ or less public if $\ell_g < \mathsf{lbl}(c)$. From the perspective of the target users who see $O_{\geq \ell_g}$, the former is achieved by including an axiom set IAS to their ontology and the latter by removing an axiom set RAS from other user's ontologies. The definition of an IAS (RAS) is a generalization of the definition of a MinA (diagnosis) [10].

Definition 3 (IAS,RAS). *A* minimal inserted axiom set *(IAS) for ℓ_g is a subset $I \subseteq O_{\ngeq \ell_g}$ such that $O_{\geq \ell_g} \cup I \models c$ and for every $I' \subset I : O_{\geq \ell_g} \cup I' \not\models c$. A* minimal removed axiom set *(RAS) for ℓ_g is a subset $R \subseteq O_{\nleq \ell_g}$ such that $O_{\nleq \ell_g} \setminus R \not\models c$ and for every $R' \subset R : O_{\nleq \ell_g} \setminus R' \models c$.*

A CS is either an IAS, a RAS, or union of both. As elaborated in [10], computing IAS and RAS is tightly related to computing explanations (also called MinA) and diagnoses. The computation by a Hitting Set Tree (HST) algorithm [11] is repeated here only briefly. The HST algorithm makes repeated calls to an auxiliary procedure that computes one CS. A tree is built, where each node is labeled with a CS and each edge with an axiom. If the CS labeling a node has n axioms ($S := \{a_1, \ldots, a_n\}$), then this node is expanded with n children: the edge to the i-th child labeled with a_i, the child labeled with a CS that is not allowed to contain neither a_i nor any ancestor's edge label. This ensures that each node is labeled with a CS distinct from those of its predecessors.

HST optimizations such as *early termination* and *node reuse* avoid redundant computations and are included in current implementations. Another optimization is putting a *cardinality limit*, applicable when not all, but only the CS of minimal cardinality $|S|$ is of interest. Then nodes might contain partial solutions in the sense that some axioms are missing, but still the smallest CS is proven to be found [10, 9].

Example 3. We continue Example 2. Assume we want to make c_1 as private as possible, i.e. $G = \{(c_1, \ell_0)\}$. All RAS are $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$, so the smallest MCS is $M_1 = \{(a_1, \ell_0)\}$ and we get $\mathsf{lbl}_{M_1}(c_1) = \ell_0$. As second example assume we want to make c_2 as public as possible, i.e. $G = \{(c_2, \ell_1)\}$. All IAS are $\{a_2\}, \{a_3\}$, so one of the smallest MCS is $M_2 = \{(a_3, \ell_1)\}$ and we get $\mathsf{lbl}_{M_2}(c_2) = \ell_1$.

4.2 Computing a Multiple Change Set for Multiple Goal Labels

An MCS for several goals consists of CS for each of the individual goals. However, it is no solution to compute single CS and combine them since this might not yield the smallest MCS or they might even conflict.

Example 4. We combine both goals of Example 3 simultaneously, i.e. we want to make c_1 as private as possible and c_2 as public as possible, $G = \{(c_1, \ell_0), (c_2, \ell_1)\}$. Just concatenating the above mentioned MCS to $M = M_1 \cup M_2 = \{(a_1, \ell_0), (a_3, \ell_1)\}$ is no MCS since $\mathsf{lbl}_M(c_2) = \ell_0 \neq \ell_1$. However, $M = \{(a_4, \ell_0), (a_5, \ell_0), (a_2, \ell_1)\}$ is an MCS.

For this reason we call any combination of CS a candidate MCS (cMCS). To compute the shortest MCS, we introduce Algorithm 2 which is similar to the HST algorithm for computing the shortest CS in [9]. The only difference is that each call to the auxiliary procedure computes a (partial) cMCS instead of a (partial) CS which is assigned to a node in the search tree, and edges are not labeled with an axiom but with a tuple (a, ℓ) which is not allowed in the child node's (partial) cMCS.

A (partial) cMCS is computed by a call $\mathsf{extract\text{-}partial\text{-}cMCS}(K, n)$ to the auxiliary procedure in Algorithm 1, where K is the set of prohibited label changes, i.e. all tuples at edges to ancestors in the HST, and n is the size of the currently known shortest MCS. The procedure comes with 2 optimizations: *CS reuse* and *cardinality limit*. As any cMCS is a combination of CS, one CS might be contained in several cMCS. Instead of computing it anew for every cMCS, the first optimization reuses it. Putting a cardinality limit is a second optimization which computes a cMCS or stops once this has reached a size n and returns a potentially partial cMCS. Computing partial CS for one goal turned out to reduce execution time [9]. In a partial cMCS, the last contained CS is partial. Partial CS are not reused.

Turning to Algorithm 2, whenever a cMCS M is found with $|M| < n$, it is shorter than our currently known shortest MCS and we can be sure that it is not partial. The question remains if it is a MCS or only a cMCS, which is checked in Line 6: neither is an

Algorithm 1 Extract cMCS with optimizations CS reuse (switch off: remove Line 11) and cardinality limit (switch off: in Line 7 replace "$n - |M|$" by "∞")

```
Procedure init-cMCS-extraction($O, \mathsf{lab}, (L, \leq), G$)
Input: $O, \mathsf{lab}$: labeled ontology; $(L, \leq)$: lattice; $G$: goal set
 1: Global: $O, \mathsf{lab}, G' := \{(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \mid (c, \ell_g) \in G,$
    $\mathrm{is}_I := \ell_g \not< \mathsf{lbl}(c) \wedge O_{\geq \ell_g} \not\models c,$        (decision to compute IAS)
    $\mathrm{is}_R := \ell_g \not> \mathsf{lbl}(c) \wedge O_{\not\leq \ell_g} \models c,$        (decision to compute RAS)
    $C_S := \emptyset\}$                                                        (reuse set for CS)
Procedure extract-partial-cMCS($K, n$)
Input: $K$: prohibited label changes; $n$: cardinality limit
Output: first $n$ elements of a cMCS
 1: $M := \emptyset$
 2: for each goal $(c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, C_S) \in G'$ do
 3:     $H := \{a \mid (a, \ell_g) \in K\}$          (set of axioms not allowed to be labelled with $\ell_g$)
 4:     if $\exists S' \in C_S : \emptyset = S' \cap H$ then
 5:         $S := S'$                                                           (CS reuse)
 6:     else
 7:         $S :=$extract-partial-CS($O, \mathsf{lab}, c, \ell_g, \mathrm{is}_I, \mathrm{is}_R, H, n - |M|$)          (defined by [9])
 8:         if $\emptyset = S$ then
 9:             return $\emptyset$        (HST normal termination for one goal fires for complete goal set)
10:         if $|S| \neq n - |M|$ then
11:             $C_S := C_S \cup \{S\}$                             (remember only non-partial CS)
12:     $M := M \cup \{(a, \ell_g) \mid a \in S\}$
13: return $M$
```

axiom allowed to have multiple labels assigned (*syntactic conflict*) nor might a change set for one goal influence any other goal which is the case if any computed boundary does not equal the goal label (*semantic conflict)*. Only after passing both checks, we update our globally known shortest known MCS M_{min} in Line 7. Loosening the constraints of a goal set, the semantic conflicts can be resolved in Line 10 or syntactic conflicts can be resolved in Line 12 which is explained in the next section.

We now show correctness of both optimizations, CS reuse and cardinality limit. Reuse of CS is correct, since the only non-constant parameter to extract a CS in Line 7 is the set of prohibited axioms H and Line 4 ensures H and the reused CS are disjoint.

Theorem 1 (Cardinality Limit Optimization). *Let O,*lab *be a labeled ontology and G a goal set. If m is the minimum cardinality of all MCS for G, the HST Algorithm 2 outputs a MCS M such that $|M| = m$.*

Proof. The described algorithm outputs a MCS since the globally stored and finally returned M_{min} is only modified when the output of extract-partial-cMCS has size strictly smaller than the limit n, has neither any syntactic nor any semantic conflict and hence only when this is indeed a MCS itself. Suppose now that the output MCS M_{min} is such that $m < |M_{\text{min}}|$, and let M_0 be a MCS such that $|M_0| = m$, which exists by assumption. Then, every MCS, i.e. every cMCS free of syntactic and semantic conflicts, obtained by calls to extract-partial-cMCS has size strictly greater than m, since otherwise, M_{min} and n would be updated. Consider now an arbitrary

Algorithm 2 HST algorithm to find smallest MCS for G

```
Procedure hst-extract-smallest-MCS($O, \mathsf{lab}, (L, \leq), G, K$)
Input: $O, \mathsf{lab}$: labeled ontology; $(L, \leq)$: lattice; $G$: goal set; $K$: prohibited label changes
Output: MCS of minimum cardinality
Global $M_{\min} := \emptyset, n := \infty, G$
init-cMCS-extraction($O, \mathsf{lab}, (L, \leq), G$)
expand-hst-MCS($K$)
return $M_{\min}$

Procedure expand-hst-MCS($K$)
Input: $K$: prohibited label changes
Side effects: modifications to $M_{\min}$ and $n$
$M :=$ extract-partial-cMCS($K, n$)
if $M = \emptyset$ then
    return                                                  (HST normal termination)
if $|M| < n$ then
    if $(a, \ell_1), (a, \ell_2) \in M \implies \ell_1 = \ell_2$ then
        if $\forall (c, \ell_g) \in G : \mathsf{lbl}_M(c) = \ell_g$ then
            $M_{\min} := M$
            $n := |M_{\min}|$
        else
            ...                                          (semantic conflict resolution)
    else
        ...                                              (syntactic conflict resolution)
for the first $(n - 1)$ label changes $(a, \ell) \in M$ do
    expand-hst-MCS($K \cup \{(a, \ell)\}$)
```

MCS M' found during the execution through a call to extract-partial-cMCS, and let $M'_n := \{(a_1, \ell_1), \ldots, (a_n, \ell_n)\}$ be the first n assignments of M'. Since M' is a (partial) MCS, it must be the case that $M_0 \not\subseteq M'_n$ since every returned MCS is minimal in the sense that no label change might be removed to obtain another MCS. Then, there must be an $i, 1 \leq i \leq n$ such that $(a_i, \ell_i) \notin M_0$. But then, M_0 will still be a MCS (and a cMCS anyway) after label change $\{(a_i, \ell_i)\}$ has been removed. Since this argument is true for all nodes, it is in particular true for all leaf nodes, but then they should not be leaf nodes, since a new cMCS, namely M_0 can still be found by expanding the HST, which contradicts the fact that $M_{\min}$ is the output of the algorithm. □

4.3 Conflict resolution

We already elaborated on syntactic and semantic conflicts which might prevent a cMCS from being a MCS. It might be the case that for a goal set, no MCS can be found.

Example 5. We continue Example 2. Assume $G = \{(c_1, \ell_4), (c_2, \ell_3)\}$. For the goal (c_1, ℓ_4) all IAS are $\{a_2\}, \{a_3\}$. For the goal (c_2, ℓ_3) all RAS are $\{a_1\}, \{a_2\}$. The cMCS $M_1 = \{(a_2, \ell_4), (a_2, \ell_3)\}$ is obviously no MCS due to a syntactic conflict. But also the remaining cMCS $M_2 = \{(a_2, \ell_4), (a_1, \ell_3)\}, M_3 = \{(a_3, \ell_4), (a_1, \ell_3)\}, M_4 = \{(a_3, \ell_4), (a_2, \ell_3)\}$ are no MCS due to semantic conflicts, since $\mathsf{lbl}_{M_2}(c_1) = \mathsf{lbl}_{M_3}(c_1) = \ell_3 \neq \ell_4$ and $\mathsf{lbl}_{M_4}(c_2) = \ell_4 \neq \ell_3$.

For these cases we introduce a generalization of an MCS called *Relaxed MCS* (RMCS) where the goal set is only partially satisfied according to a defined strategy. For the special case of no conflict, the RMCS equals the MCS. We identified 4 strategies to resolve conflicts, where we focus on syntactic conflict resolution only:

1. Overrestrictive: accept lower labels for a minimal number of consequences than specified by the goal label. Formally, $\forall(c,\ell_g)\in G : \mathsf{lbl}_M(c)\neq\ell_g \implies \mathsf{lbl}_M(c)<\ell_g$ and cardinality $|\{(c,\ell_g)\in G \mid \mathsf{lbl}_M(c)\neq\ell_g\}|$ is minimal. Applied to the above example, $\{(a_2,\ell_3)\}$ is a RMCS.
2. Overpermissive: accept higher labels for a minimal number of consequences than specified by the goal label. Formally, $\forall(c,\ell_g)\in G : \mathsf{lbl}_M(c)\neq\ell_g \implies \mathsf{lbl}_M(c)>\ell_g$ and cardinality $|\{(c,\ell_g)\in G \mid \mathsf{lbl}_M(c)\neq\ell_g\}|$ is minimal. Applied to the above example, $\{(a_2,\ell_4)\}$ is a RMCS.
3. Override strategy: The goal G set is split up into fragments G_i so that $G = G_1\cup\ldots\cup G_n$ for which individual MCS M_i can be computed. The changed label assignment $((\mathsf{lab}_{M_1})\ldots)_{M_n}$ is obtained by sequentially applying each MCS M_i, where the order can be chosen based on some prioritization. This implies that labels changed by one MCS might be changed again by any subsequent MCS. Applied to the above example, splitting up G into G_1 and G_2, $G_1=\{(c_1,\ell_4)\}$ yields MCS $M_5=\{(a_2,\ell_4)\}$, subsequently $G_2=\{(c_2,\ell_3)\}$ yields MCS $M_6=\{(a_2,\ell_3)\}$.

Strategy 3 although easy to implement has an unacceptable drawback, conflicting our RMCS definition: even if there is a MCS for the union of all goal subsets, a sequentially applied MCS for one goal subset might override a previous for another goal subset since they are computed independently of each other. For this reason we focus on strategies 1 and 2 for resolution of syntactic conflicts.

Algorithm 3 describes the resolution of syntactic conflicts. It is an adapted version of Algorithm 2, where additionally the global variable r stores the minimal number of overpermissive (overrestrictive) consequence labels and N stores the RMCS with minimal r. Again this Algorithm relies on the cMCS extraction Algorithm 1 and the optimization of reusing CS can be applied. The cardinality limit optimization is of no use here since if no MCS is found, then no cardinality limit is set and the HST is fully expanded.

There are goal sets yielding semantic conflicts but no syntactic conflicts in cMCS. These are not solved by syntactic conflict resolution. For these cases not only IAS and RAS, but complete explanations and diagnoses need to be taken into account, as the following example shows.

Example 6. We continue Example 2. Assume $G=\{(c_1,\ell_2),(c_2,\ell_5)\}$. For the goal (c_1,ℓ_2) all IAS are $\{a_4\},\{a_5\}$. For the goal (c_2,ℓ_5) all IAS are $\{a_2\},\{a_3\}$, all RAS are $\{a_1\},\{a_2,a_3\}$. Obviously no combination of CS for both goals yields a syntactic conflict. Nevertheless there is no MCS since every combination of CS has a semantic conflict. After conflict resolution, an overpermissive RMCS is $N_{OP}=\{(a_4,\ell_2),(a_2,\ell_2\oplus\ell_5=\ell_1),(a_3,\ell_5\}$, yielding $\mathsf{lbl}_{N_{OP}}(c_1)=\ell_1,\mathsf{lbl}_{N_{OP}}(c_2)=\ell_1$. An overrestrictive RMCS is $N_{OR}=\{(a_4,\ell_2),(a_2,\ell_2\otimes\ell_5=\ell_0),(a_3,\ell_5)\}$, yielding $\mathsf{lbl}_{N_{OR}}(c_1)=\ell_5,\mathsf{lbl}_{N_{OR}}(c_2)=\ell_5$.

Algorithm 3 Computing a RMCS, overpermissive strategy (for overrestrictive strategy: replace "$\oplus$" with "$\otimes$" in Line 3, "$\geq$" with "$\leq$" in Line 4, "$>$" with "$<$" in Line 5)

Basis is the Algorithm 2. In Procedure hst-extract-smallest-MCS, add global variables $N := \emptyset, r := \infty$, and add before Line 4:

```
1: if $\emptyset = M_{\text{min}}$ then
2:     return $N$
```

In Procedure expand-hst-MCS, replace Line 12 for syntactic conflict resolution with:

```
1: $N' := M$
2: for each $a : (a, \ell_1), (a, \ell_2) \in N' \wedge \ell_1 \neq \ell_2$ do
3:     $N' := N' \setminus \{(a, \ell_1), (a, \ell_2)\} \cup \{(a, \ell_1 \oplus \ell_2)\}$
4: if $\forall (c, \ell_g) \in G : \mathsf{lbl}_{N'}(c) \geq \ell_g$ then          (fulfills overpermissive strategy)
5:     $r' := |\{(c, \ell_g) \in G \mid \mathsf{lbl}_{N'}(c) > \ell_g\}|$
6:     if $r' < r$ then
7:         $N := N'$
8:         $r := r'$
```

5 Experiments

We implemented and evaluated our algorithms empirically with large practical ontologies. The following sections describe our test setting and the results.

5.1 Test Procedure and Test Data

We test on a PC with 2GB RAM and Intel Core Duo CPU 3.16GHz. We implemented all approaches with Java 1.6, Pellet 2.0 and OWL API trunk revision 1150. As labeling lattice $(L, \leq)$ we use the one introduced in Figure 1. We use the top lattice element ℓ_1 for public knowledge, ℓ_2 for intermediate knowledge and ℓ_3 for top secret knowledge.

Our test ontologies O^{GEOM}[1], O^{MGED}[2], O^{PROCESS}[3] are selected ontologies from the TONES Ontology Repository[4] with a high number of individuals. At time of their download on March 25th 2010, they had the characteristics given in Table 1. The test ontology O^{FUNCT} is an OWL ontology for functional description of mechanical engineering solutions presented in [6].

In a first experimental setting we tested the availability of access control by query rewriting vs. access control by axiom filtering. Initially each ontology axiom is labeled ℓ_1 so that the complete ontology is public. This reflects default allow-all behavior of a security policy. Then for each concept C in the ontology, we apply access restriction $AR = C(x)$ by including each query result $c = \mu(AR)$ with goal label ℓ_3 in the goal set. The computed MCS is used to create a newly labeled ontology, on which we perform the following queries. We count for every C-instance the instance relationships to concepts other than C which are available for public users (ℓ_1). With query rewriting

[1] http://i2geo.net/ontologies/dev/ontology.owl

[2] http://mged.sourceforge.net/ontologies/MGEDOntology.owl

[3] http://sweet.jpl.nasa.gov/ontology/process.owl

[4] owl.cs.manchester.ac.uk/repository/

Ontology	DL expressivity	♯logical	♯concepts	♯individuals	♯goal sets	♯goals per
		axioms				goal set
$O^{\textsc{Funct}}$	$\mathcal{ALCOIN}(\mathbf{D})$	3189	115	545	102	12.2
$O^{\textsc{Geom}}$	$\mathcal{ALCHOIN}(\mathbf{D})$	8803	589	2010	571	14.1
$O^{\textsc{Process}}$	$\mathcal{ALCHOF}(\mathbf{D})$	2578	1537	150	40	20.9
O^{MGED}	$\mathcal{ALEOF}(\mathbf{D})$	1387	234	681	125	28.8

Table 1. Test sets consisting of ontologies and goal sets

their count is 0. With axiom filtering their count is the availability gain of axiom filtering vs. query rewriting. For cMCS extraction defined by Algorithm 1, we tested both optimizations CS reuse and cardinality limit separately and their combination. In this setting every cMCS is automatically an MCS since there are no conflicting goals. Although not included in Algorithm 2 for transparency reasons, the mentioned usual HST optimizations *early termination* and *node reuse* are included in our implementation.

In a second experimental setting we tested conflict resolution strategies in cases where multiple goals conflict each other so that no MCS can be computed without relaxing one of the goals. We test the overrestrictive conflict resolution approach vs. the overpermissive conflict resolution approach of Algorithm 3 with the same ontologies. Only the CS reuse optimization of the auxiliary procedure in Algorithm 1 to extract cMCS is used, cardinality limit is not used for reasons explained in Section 4.3. First all axioms are labeled with intermediate security level, i.e. ℓ_2. A goal set is created for each concept C containing the same consequences described above, but now one half of this set has goal label ℓ_1 and the other half ℓ_3. Some of the resulting goal sets are contradictory. We test Algorithm 3 to compute a RMCS with overpermissive vs. overrestrictive conflict resolution strategy for the same goal set and we count the number of overpermissive/overrestrictive consequence labels.

For both experiments the test data characteristics are given in Table 1. The number of goal sets and of goals per goal set are the same for both experiments since they contain the assertions to each of the ontology's concepts, only with different goal labels. In order to limit runtime we compute in maximum 10 cMCS before the HST Algorithms 2 and 3 return, so there might be MCS or RMCS of lower cardinality.

5.2 Empirical Results

The experimental results for the first experiment are given in Table 2. It compares availability of access control by query rewriting vs. access control by axiom filtering and it compares performance of both optimizations cardinality limit vs. CS reuse. The given total number of CS includes reused CS. The number of cMCS is equal to the number of MCS since the goals contain no conflicts with the first experiment. The number of gained assertions confirms that our ideas improve availability of knowledge when using axiom filtering instead of query rewriting. While the number of gained assertions is comparable between the optimizations applied, their runtime differs significantly. CS reuse alone, and also in combination with cardinality limit runs significantly faster com-

Test set	optimization	Results (averages per goal set)					
		♯CS	♯reused CS	♯cMCS	|MCS|	runtime	♯gained
				= ♯MCS		(minutes)	assertions
O^{Funct}	card. limit	131.8	0.0	3.9	23.9	3.6	28.5
	CS reuse	135.2	118.4	3.9	24.0	0.7	28.6
	both	132.6	115.7	3.9	24.1	0.6	28.4
O^{Geom}	card. limit	146.9	0.0	2.6	9.2	24.0	43.4
	CS reuse	148.9	132.9	2.5	9.3	4.2	43.3
	both	147.3	131.1	2.6	9.3	4.2	43.3
O^{Process}	card. limit	199.3	0.0	6.9	12.0	2.3	92.6
	CS reuse	250.9	217.8	6.7	12.2	0.6	91.8
	both	197.9	165.0	6.8	12.2	0.6	91.9
O^{MGED}	card. limit	n/a	n/a	n/a	n/a	n/a	n/a
	CS reuse	286.4	253.4	2.9	15.1	115.9	53.9
	both	265.1	232.4	3.0	15.1	114.3	54.1

Table 2. Gained assertions compared to query rewriting, performance of optimizations

pared to using cardinality limit optimization only. Testing O^{MGED} with cardinality limit optimization did not terminate after 4 days, so no results are provided.

The experimental results for the second experiment comparing conflict resolution with overrestrictive strategy vs. overpermissive strategy are given in Table 3. Only some of the goal sets constructed as described above are conflicting, and results are only given for those. Only the given percentage of the goals in one goal set are enforced, the remaining consequences have overpermissive/overrestrictive labels making them more public/private than intended by the goal set. The runtime limit of 10 cMCS was hit in every case, making the HST algorithm stop so there might be RMCS with less overpermissive/overrestrictive consequence labels when relaxing this runtime limit.

Test set	♯goal	♯goals	strat-	Results (averages per conflicting goal set)					
	sets	per	egy	♯cMCS	♯RMCS	|RMCS|	runtime	♯OR/OP	% of
	confl.	confl.					(min-	cons.	enforced
		goal set					utes)	labs	goals
O^{Funct}	19	50.3	OR	10.0	10.0	101.4	2.2	19.5	61%
			OP	10.0	10.0	110.0	2.0	20.3	60%
O^{Geom}	39	150.7	OR	10.0	10.0	139.4	45.4	63.3	58%
			OP	10.0	10.0	140.4	37.0	52.1	65%
O^{Process}	23	31.0	OR	10.0	10.0	32.3	0.9	12.7	59%
			OP	10.0	10.0	32.6	0.8	11.0	64%
O^{MGED}	16	165.8	OR	10.0	10.0	140.4	814.6	75.6	54%
			OP	10.0	10.0	141.6	780.8	51.9	69%

Table 3. Conflict resolution with overrestrictive (OR) strategy vs. overpermissive (OP) strategy

6 Conclusions

We considered scenarios where different parts of a given ontology should be visible for different users. We introduced access restrictions intentionally defined by means of a query. The answer to that query is the set of those axioms and consequences of the ontology, which have to be hidden. We compared two basic approaches to enforce those access restrictions: query rewriting and axiom filtering. Compared to query rewriting, axiom filtering allows higher availability in the sense of more answers delivered to a user without unveiling any secret.

Axiom filtering relies on a axiom labeling. The problem solved by this paper is to find an optimal axiom labeling to enforce given access restrictions. Given a query-generated goal set containing consequences and intended labels, our algorithms compute a minimal change set defining a new axiom labeling. We show that a change set does not always exist since a goal set might contain conflicts, and we provide 2 conflict resolution strategies to relax the goal set so that a change set can be computed. Our experimental results show that our algorithms behave well in practical scenarios.

As future work we will look at other criteria for the minimality of change sets for example not counting the amount of changed axiom labels but the distance of the new from the old label in the lattice, the amount of other consequence's labels changed, or the amount of affected users. We will also look at resolution of semantic conflicts and study a more expressive goal language to define for each single goal of a goal set whether it may be lowered or lifted in case of conflicts.

References

1. F. Baader, M. Knechtel, and R. Peñaloza. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In *Proc. of ISWC 2009*, volume 5823 of *LNCS*, pages 49–64, 2009.
2. F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 20(1):5–34, 2010. Special Issue: Tableaux and Analytic Proof Methods.
3. D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. View-based query answering over description logic ontologies. In *Proc. of KR 2008*, 2008.
4. W. Chen and H. Stuckenschmidt. A model-driven approach to enable access control for ontologies. In *Proc. of WI 2009*, pages 663–672, 2009.
5. C. Farkas and S. Jajodia. The inference problem: a survey. *SIGKDD Explor. Newsl.*, 4(2):6–11, 2002.
6. A. Gaag, A. Kohn, and U. Lindemann. Function-based solution retrieval and semantic search in mechanical engineering. In *Proc. of ICED 09*, 2009.
7. B. C. Grau and I. Horrocks. Privacy-preserving query answering in logic-based information systems. In *Proc. of ECAI-2008*, 2008.
8. A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proc. of ISWC/ASWC 2007*, volume 4825 of *LNCS*, pages 267–280, 2007.
9. M. Knechtel and R. Peñaloza. Correcting access restrictions to a consequence. In *Proc. of DL 2010*, volume 573 of *CEUR-WS*, 2010.
10. M. Knechtel and R. Peñaloza. A generic approach for correcting access restrictions to a consequence. In *Proc. of ESWC 2010*, volume 6088 of *LNCS*, pages 167–182, 2010.
11. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
12. E. Sirin and B. Parsia. SPARQL-DL: SPARQL queries for OWL-DL. In *Proc. of OWLED 2007*, 2007.