RBAC AUTHORIZATION DECISION WITH DL REASONING

Martin Knechtel, Jan Hladik
SAP AG, SAP Research CEC Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany

ABSTRACT

Access control is crucial also for the Semantic Web. Technologies and Standards from the Semantic Web Community itself provide powerful means to model access control definitions and automatically reason about them. We extend Hierarchical Role Based Access Control by a class hierarchy of the accessed objects and give it the name RBAC-CH. We present a concept to implement this model in a DL knowledge base in the form of an OWL 1.1 ontology. The permissions are defined for user roles on object classes. The concrete permissions of users to objects are then automatically derived by a reasoning service. We present a straightforward ontology model and evaluate it in a running example with a state of the art reasoner. For the RBAC policy enforcement we need to run the reasoner only once and at runtime we only need to read out the inferred knowledge base to decide about authorization.

KEYWORDS

RBAC, RBAC-CH, object class hierarchy, Description Logic, OWL 1.1, SROIQ

1. INTRODUCTION AND MOTIVATION

Role Based Access Control (RBAC) (Sandhu et al., 2000) is a standardized model to indirectly assign permissions to users by user roles. The permissions of users are the sum of all permissions of their assigned roles. In enterprise context the user roles often represent job descriptions of employees. The roles build a hierarchy along which permissions are inherited. We follow the proposal of (Chae and Shiri, 2007) to introduce a hierarchy of object classes along which permissions are inherited in addition to the hierarchy of user roles. In file systems the inheritance of permissions along the directory tree has been established for a long time. If we want to control access on other hierarchical data structures, like a subsumption hierarchy of ontology concepts, then permission inheritance along the hierarchy seems natural.

Different formalizations are suitable for RBAC, especially Description Logics has been proposed multiple times in literature. Description Logic (DL) (Baader et al., 2007) systems provide their users with inference services that deduce implicit knowledge from the explicitly represented knowledge, which can be employed to infer permissions of users to objects from a definition of permissions of user roles on object classes. For these inference services to be feasible, the underlying inference problems must at least be decidable, since DL is a decidable fragment of First Order Logic this is provided.

The proposal by (Chae and Shiri, 2007) is based on DL but has several flaws. Their object class hierarchy is inverted, which leads to unintended inferences, furthermore DL semantics is not respected and their running example provides wrong results. A detailed discussion follows in the related work section. We focus on their paper and reuse their example scenario in order to compare our results to their approach. This paper targets to fix the flaws of (Chae and Shiri, 2007).

The paper is structured as follows. We will give a short introduction to Hierarchical RBAC and our extension. We will then discuss related work and point out weaknesses. Then we will introduce OWL 1.1 and the DL $\mathcal{SROIQ}$ and explain why we need it. With these prerequisites, we will present our ontology for RBAC-CH, followed by a running example. We conclude our results and point to future research.

2. EXTENDING HIERARCHICAL RBAC TO RBAC-CH

Hierarchical Role Based Access Control inserts an indirection between users and their permissions on objects. Sandhu, Ferraiolo and Kuhn are the authors of the original proposal (Sandhu et al., 2000) for the NIST standard[1]. Users (*U*) are assigned to roles ($\mathcal{R}$) by the user assignment (*UA*) relation. Roles form a hierarchy by a partial order (*RH*). The hierarchy can be an inheritance hierarchy, s.t. the assignment of a role to a user implies the assignment of all junior roles. Roles get permissions assigned by the permission assignment (*PA*) relation. For example in a file system, user *Edward* can execute the file *start.bat* since he is assigned to the role *Operating System Developer* which inherits the permission from the super role *Remote Client* to execute *start.bat*.

We follow the proposal of Chae and Shiri to abstract from individual objects to object classes (Chae and Shiri, 2007). In the resulting access control model, permissions are not assigned for user roles on objects but on object classes. We remove permission (*P*) and introduce objects (*O*), object classes ($\mathcal{C}$), and object class hierarchy (*CH*). Additionally we introduce actions (*A*), which are missing in (Chae and Shiri, 2007). We replace permission assignment (*PA*) with a ternary relation to model allowed actions of roles on object classes. Each object (*O*) is classified with an object class by the object assignment relation (*OA*). Object classes form a hierarchy by a partial order (*CH*). We give this extended RBAC version the name Hierarchical Role Based Access Control with Object Class Hierarchy (RBAC-CH) and define it formally by Def. 1, the symbol notation adheres to (Shandu et al.,2000).

Def. 1 (RBAC-CH) *The Hierarchical Role Based Access Control with Object Class Hierarchy is defined by the following constituents.*

– *U,A and O are the sets of users, actions and objects*
– *$\mathcal{R}$ and $\mathcal{C}$ are the sets of user roles and object classes*
– *$u \in U$, $a \in A$, $o \in O$, $r \in \mathcal{R}$, $c \in \mathcal{C}$ denote individual user, action, object, role and object class*
– *$UA \subseteq U \times \mathcal{R}$ is a many-to-many assignment relation of users to roles, i.e. user $u \in U$ has role $r \in \mathcal{R}$ assigned if $(u,r) \in UA$*
– *$OA \subseteq O \times \mathcal{C}$ is a many-to-many assignment relation of objects to object classes, i.e. object $o \in O$ is classified with $c \in \mathcal{C}$ if $(o,c) \in OA$*
– *$PA \subseteq \mathcal{R} \times A \times \mathcal{C}$ is a many-to-many-to-many assignment relation of roles to object classes via allowed actions, i.e. the role $r \in \mathcal{R}$ has the permission for action $a \in A$ on object class $c \in \mathcal{C}$ if $(r,a,c) \in PA$*
– *$RH \subseteq \mathcal{R} \times \mathcal{R}$ is a partial order on $\mathcal{R}$ called the role hierarchy or role dominance relation also written as $\geq R$*
– *$CH \subseteq \mathcal{C} \times \mathcal{C}$ is a partial order on $\mathcal{C}$ called the object class hierarchy or object class dominance relation also written as $\geq C$*

The idea is that person $u \in U$ can perform action $a \in A$ on object $o \in O$ if the role $\mathcal{R}$ of the user has the explicit permission for action a on the class $\mathcal{C}$ which classifies object o. Our example above is now changed so that *Edward* can execute *start.bat* because it is classified as *Program File* and all *Remote Clients* can execute all *Executable Files* which is the superclass of *Program File*. Since we only allow positive authorizations, multiple roles assigned to one user cannot imply conflicting permissions. In the next section we discuss existing approaches to formalize access control models with our focus on RBAC and DL.

3. RELATED WORK

Literature provides a wide range of models to formalize user role hierarchies and access policies with Semantic Web technologies. In (Bonatti and Olmedilla, 2007) the policy languages KAOS and REI which use DL reasoning and PeerTrust and Protune which use logic programming (LP) reasoning are discussed.

[1] The current standard can be accessed at the NIST Webpage http://csrc.nist.gov/rbac/.

Approaches based on DL (Chae and Shiri, 2007, Kolovski et al., 2007, Zhao et al., 2005, Finin et al., 2008) turned out to be useful since important policy analysis and enforcement tasks can be reduced to standard reasoning services. None of them is intended to be used at runtime and is based on the realization reasoning service like ours. To the best of our knowledge, our concept is the only DL implementation of RBAC focusing on runtime.

Kolovski et al. present a DL approach to analyze XACML policies (Kolovski et al., 2007). The DL model of the access policy also introduces negative authorization and a decision mechanism for conflicting policy rules based on Defeasible Description Logics (DDL). An important difference to our work is that they focus not on runtime but design time to audit a policy in order to find mistakes and inconsistencies, since the services used are computationally expensive. For each authorization decision the reasoner needs to rerun in their approach, whereas in our approach classification and realization runs only once. Authorization decisions can then be enforced without rerunning the reasoner.

The idea for an object class hierarchy in RBAC was introduced by (Chae and Shiri, 2007), however they applied essential properties of DL in a wrong way, did not respect DL semantics, did not use ABox assertions correctly, miss a discussion of the open world assumption and get wrong results with their running example. In the following we discuss some details.

- *The authors apply essential properties of DL in a wrong way and do not respect DL semantics.* They invert the object class hierarchy and use the existential quantifier $\exists$ which references only some individuals of a concept although their intension is to reference all individuals of a concept. In order to say "all system administrators can execute all files" they model $SysAdmin \sqsubseteq \exists canExecute.File$ which means "a system administrator can execute some file". Since their inverted object class hierarchy contains $File \sqsubseteq ExeFile$, it follows implicitly that $SysAdmin \sqsubseteq \exists canExecute.ExeFile$. For a manager who is only allowed to execute executable files, i.e. $Mag \sqsubseteq \exists canExecute.ExeFile$ it does not follow implicitly that $Mag \sqsubseteq \exists canExecute.File$, which would mean that a manager could execute arbitrary files. Thus, the authors reach the intended behavior for the inheritance of access rights with respect to a hierarchy of object classes, but they miss DL semantics. The model of their ontology does not follow DL intuition and further inferences apart from the intended ones are dubious. For example the object class hierarchy is modeled "inverse", e.g. $File \sqsubseteq ExeFile$, which has the unwanted meaning that every file is an executable file which is obviously not the case.
- *The authors do not use ABox assertions correctly.* DL offers a natural way to assign individuals from the ABox to concepts of the TBox in order to classify individuals, but this mechanism is not used. Instead of assigning objects to object classes and users to roles by ABox assignments they introduce object properties *assign* and *classify*. Since only pairs of individuals can be asserted to object properties and not pairs of one individual and one concept, they introduce atomic concepts with exactly one individual to connect ABox and TBox, which is unnecessarily complicated and we show that ABox assertions are the suitable alternative.
- *The authors miss a discussion of the open world assumption.* They do not define how to treat incomplete knowledge which is inherent to DL knowledge bases, which makes it unclear whether a missing permission is equivalent to a prohibition or is just unknown. In our paper we decide explicitly for one option.
- *The authors get wrong results with their running example.* They present a running example which seems not to work correctly, since in their access matrix, the role *SysAdmin* has execution permission on *File*, but not on all of its subconcepts like *SysFile*, *ConFile*, *LocFile* and *ElcJ*. For comparison reasons in our paper we reuse the user role and object class hierarchy as well as the permission assignments of the authors. There are more mistakes in their inferred access control matrix, which we summarize in our later comparison in Tab. 4. As we show in our paper, DL can be applied to model RBAC with an object class hierarchy and get the right authorization results.

The DL representation of RBAC by (Zhao et al., 2005) tied permissions to objects which results in an explosion of the number of permissions. Furthermore they do not consider an object class hierarchy.

Also (Finin et al., 2008) present an RBAC implementation in OWL, but they do not introduce an object class hierarchy. Apart from that, several design decisions are questionable. While they do not make explicit which specific DL language from the whole DL language family they chose, they state that OWL is used. Assuming that they use $\mathcal{SHIF(D)}$, which is the basis of the smallest OWL dialect OWL Lite, some design decisions are still unclear. To build a hierarchy of properties they use a workaround which is not necessary, since the language supports subproperties. Disjoint Properties are also solved by workaround although they can be modeled with the DL $\mathcal{SROIQ}(\mathcal{D})$, which is supported by state of the art reasoners and is the candidate for the upcoming OWL 1.1 standard. In parts a consequence of their workarounds which could be avoided, they use rules in N3Logic in addition to DL subsumption reasoning. They do not focus on runtime application since for every authorization decision, a new reasoning run is needed. Furthermore they state that "queries about a general class of access requests (Can every student use lab printers?) can be answered efficiently using a standard DL reasoner through subsumption reasoning". Again this is a question of the chosen DL language from the DL language family. With OWL Lite or OWL DL which they apparently have chosen this cannot be modeled whereas(Rudolph et al., 2008) have shown that it can be done with $\mathcal{SROIQ}(\mathcal{D})$ and we apply this idea in our paper.

Lattice based access control (LBAC) uses a hierarchy of labels in a partially ordered lattice. It was originally developed for military, but is in general applicable to any situation where information flow is of concern (Shandhu, 1993). RBAC can be considered a superset of LBAC, since RBAC introduces more flexibility while it can still simulate LBAC.

We will now introduce the DL expressive enough to formalize RBAC-CH and propose our ontology.

4. OWL 1.1 AND DESCRIPTION LOGIC $\mathcal{SROIQ}$

A DL system consists of knowledge base *K* and inference engine. The knowledge base contains two components, TBox and ABox *K=<T, A>*. The TBox contains vocabulary which is used to describe a concrete world in the ABox. OWL is the Web ontology standard by the W3C (Bechhofer et al., 2004). The OWL DL species of OWL uses the DL $\mathcal{SHOIN}$(**D**) (Baader et al., 2007) as formal base. The "**D**" represents support for concrete domains, e.g. reasoning about integer or date values. The W3C Member Submission for OWL 1.1 (Patel-Schneider and Horrocks, 2006) is an extension of the OWL DL Standard and uses the DL $\mathcal{SROIQ}$(**D**) (Horrocks et al., 2006). Apart from other features, it introduces *property chain inclusion axioms*. As we show later we make use of this new feature. Concepts contained in the TBox can be built by constructors. A selection of them, the ones we need in this paper, are given in Tab. 1.

The semantics of a concept is build by means of an interpretation $I = (\Delta, \cdot^I)$. The domain Δ of I is a non-empty set and the interpretation function $\cdot^I$ that maps each concept name C to a set $C^I \subseteq \Delta$ and each property[2] name P to a binary relation $P^I \subseteq \Delta \times \Delta$. The extension of $\cdot^I$ to some complex concepts is given in Tab. 1. TBoxes are an expressive formalism that allows making statements about complex concepts. ABoxes assert individuals to concepts and pairs of individuals to properties. Sometimes it is convenient to use individual names (also called nominals) not only in the ABox but also in concept constructors. The "fills" constructor *P*:*a* is also listed in Tab. 1. Table 2 summarizes selected axioms supported by OWL 1.1 and needed in this paper.

Table 1. Some OWL 1.1 concept constructors, DL syntax and semantics

OWL 1.1 Functional Syntax	DL syntax	semantics
owl:Thing	$\top$	Δ
owl:Nothing	$\bot$	$\emptyset$
ObjectIntersectionOf(C D)	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
ObjectUnionOf(C D)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
ObjectHasValue(P a)	$P : a (= \exists P.\{a\})$	$\{x \in \Delta \mid (x, a) \in P^{\mathcal{I}}\}$
ObjectSomeValuesFrom(P C)	$\exists P.C$	$\{x \in \Delta \mid \exists y : (x, y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
ObjectAllValuesFrom(P C)	$\forall P.C$	$\{x \in \Delta \mid \forall y : (x, y) \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
ObjectComplementOf(C)	$\neg C$	$\Delta \setminus C^{\mathcal{I}}$

[2] Note that in the DL community this binary relation is often called *role* instead of *property*. We stick to the OWL terminology here in order to avoid confusion between *DL roles* and *user roles* of the access model.

Table 2. OWL 1.1 axioms needed for our approach

OWL 1.1 Functional Syntax	DL syntax	semantics
SubClassOf(C1 C2)	$C_1 \sqsubset C_2$	$C_1^{\mathcal{I}} \subset C_2^{\mathcal{I}}$
ClassAssertion(C a)	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
ObjectPropertyAssertion(P a b)	$P(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
InverseObjectProperties(P1 P2)	$P_1 \equiv P_2^-$	$P_1^{\mathcal{I}} = \{(x,y) \mid (y,x) \in P_2^{\mathcal{I}}\}$
SubObjectPropertyOf(P1 P2)	$P_1 \sqsubseteq P_2$	$P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
SubObjectPropertyOf(SubObjectPropertyChain(P1 P2) P)	$P_1 \circ P_2 \sqsubseteq P$	$\{(a,c) \in \Delta \times \Delta \mid \exists b : (a,b) \in P_1^{\mathcal{I}} \wedge (b,c) \in P_2^{\mathcal{I}}\} \subseteq P^{\mathcal{I}}$

5. OWL 1.1 ONTOLOGY FOR RBAC-CH

The permission ontology describes objects to be accessed, user roles and permissions of user roles to object classes. The concept *Role* is the superconcept of all user roles. The concept *Object* is the superconcept for all object classes. Concrete objects are instances of *Object* and its subconcepts. Concrete users are instances of *Role* and its subconcepts. Thus technically the classification of an object into an object class and the assertion of a user to a user role are realized by ABox assertions.

Following the open world assumption (OWA), the knowledge base is always incomplete. If a fact is not contained in or does not follow from the knowledge base, we cannot conclude that the complement is true. In our case some positive authorizations are contained in the knowledge base explicitly, some are contained implicitly, and some are not contained. If we check for an authorization that does not exist, the reasoner answers that it is unknown. Because we want a binary authorization decision, we decided to interpret "unknown" as a negative authorization, thus the user does not get the authorization.

Permissions are modeled by object properties. The definition of permissions for user roles on object classes has an important requirement: When permission for a role on an object class is defined, we want to model that *every* individual of a user role has an object property to *every* individual of an object class. An example is "all system administrators have access to all types of files". Obviously it is essential to model such statements in any formal approach to RBAC, but statements like these are not generally supported by OWL. However Rudolph et al. propose in (Rudolph et al., 2008) the *concept product* to express such statements, which can be simulated in the very expressive DL $\mathcal{SROIQ}$ which is likely to become standard in OWL 1.1 (Patel-Schneider and Horrocks, 2006). The concept product putting all individuals of concept *C* in relation to all individuals of concept *D* by object property *P* is written as $C \times D \sqsubseteq P$.

We provide an example, how we apply the concept product to model the general permission assignment "all system administrators read all types of files" with the concept product $SysAdmin \times File \sqsubseteq canRead$ simulated in a $\mathcal{SROIQ}$ knowledge base:

$$canRead_1 \circ canRead_2^- \sqsubseteq canRead$$

$$SysAdmin \sqsubseteq \exists canRead_1.\{a\}$$

$$File \sqsubseteq \exists canRead_2.\{a\}$$

In the inferred ABox after the realization reasoning service was performed, for every user individual we can query for object individuals related by the object property *canRead* to retrieve the Capabilities of the user. The inverse property $canBeReadBy \equiv canRead^-$ can be added to allow retrieval of an Access Control List (ACL) for each object. By querying the ABox we can construct the complete Access Control Matrix, as we show in the following section.

6. RUNNING EXAMPLE

The TBox contains the concept hierarchy with the user roles as well as the object class hierarchy as depicted in Fig. 1. In the concept definitions, we use abbreviations for the user roles *Remote Client* (RemCli), *Local Client* (LocCli), *Manager* (Mag), *Operating System Developer* (OSDev) and *System Administrator* (SysAdmin). We also use abbreviations for the file types *System File* (SysFile), *Electronic Journal* (ElcJ), *Executable File* (ExeFile), *Local File* (LocFile), *Configuration File* (ConFile), *Program File* (ProFile) and *Executable System File* (ExeSysFile). The concept hierarchies are defined explicitly in the form of General Concept Inclusions (GCIs) of the form $ExeFile \sqsubseteq File$, $ProFile \sqsubseteq ExeFile$, $SysAdmin \sqsubseteq Mag$ etc. We do not provide all GCIs explicitly, since they follow from the figure.

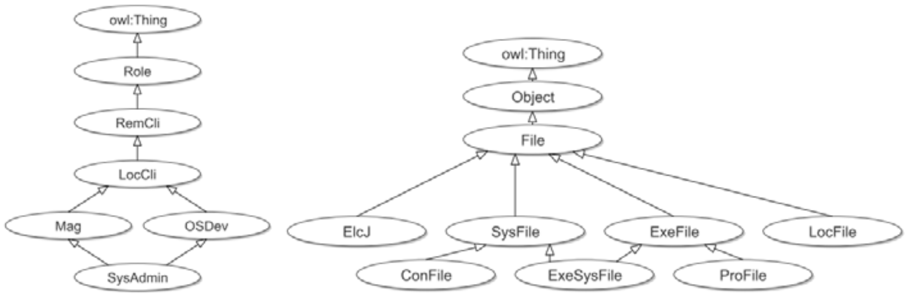

Figure 1. User role and object class hierarchy (white arrow heads: SubClassOf relation)

Furthermore the TBox contains the assignment of permissions for user roles on object classes. In the following example, we define that every remote client can execute every file which is an executable file with the concept product $RemCli \times ExeFile \sqsubseteq canExecute$. To simulate this statement in $\mathcal{SROIQ}$, the GCIs for the user role and object class hierarchy are extended as follows:

$$RemCli \sqsubseteq Role \sqcap \exists canExecute_1.\{auxIndRemCliExeFile_1\} \qquad (1)$$
$$ExeFile \sqsubseteq File \sqcap \exists canExecute_2.\{auxIndRemCliExeFile_1\} \qquad (2)$$

The TBox also contains the object properties to define the operation *canExecute* to execute a file and its inverse *canBeExecutedBy.*[3]

$$canExecute_2- \equiv canExecute_2^- \qquad (3)$$
$$canExecute_1 \circ canExecute_2- \sqsubseteq canExecute \qquad (4)$$
$$canBeExecutedBy \equiv canExecute^- \qquad (5)$$

The ABox contains assertions of users to user roles and files to object classes in the form *OSDev*(*edward*), *ProFile*(*programFile1*) etc.

After classification and realization of the knowledge base by a reasoner, we can directly read the authorizations from the inferred object properties between the ABox individuals. In our example, for the individual *edward* we find *canExecute*(*edward*,*programFile1*) and *canBeExecutedBy*(*programFile1*,*edward*). An extract of the resulting knowledge base is depicted in Fig. 2.

In the example we only defined "remote clients can execute executable files. We stick to the example scenario from (Chae and Shiri, 2007) which contains further explicit permissions, which we have given in Tab. 3. The permissions were added to our ontology similarly to our given example, whereas *x* represents the action *canExecute*, and *r* and *w* represent *canRead* and *canWrite* respectively. The reading direction to construct the concept products is $Role \times ObjectClass \sqsubseteq action$. The simulation in $\mathcal{SROIQ}$ is done as described.

[3] We explicitly needed to define *canExecute*- explicitly instead of using $canExecute^-$ for the sake of compatibility issues with Pellet 1.5. Conceptually this is not needed, but implementation specific.

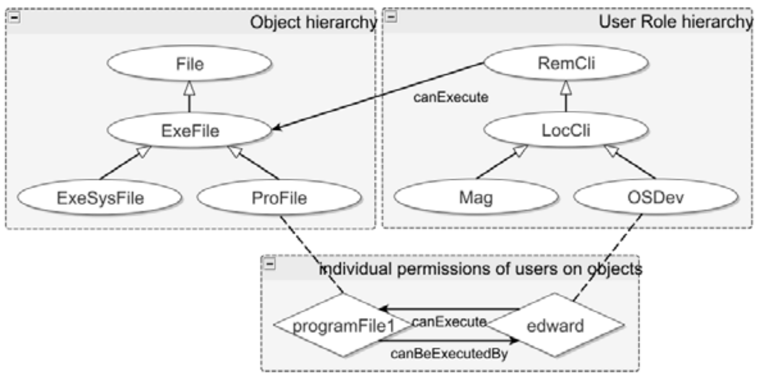

Figure 2. Infer authorization decisions from explicit permission assignments (ovals: concepts, diamonds: individuals, white arrow heads: SubClassOf relation, black arrow heads: object property, dashed line: ClassAssertion)

Due to the user role hierarchy and the object class hierarchy, from the explicit permissions in Tab. 3 implicit permissions in Tab. 4 were made explicit by DL reasoning. Our ontology provides correct results, whereas in (Chae and Shiri, 2007) some inferences are not correct which we have given in parentheses for comparison.

Table 3. Access Matrix with explicit permissions

	ElcJ	LocFile	ConFile	SysFile	ExeSysFile	ProFile	ExeFile	File
SysAdmin								r,w,x
Mag			r,w					
OSDev								
LocCli	r							
RemCli		r,w					x	

Table 4. Access Matrix with explicit and implied permissions, incorrect solutions in parentheses

	ElcJ	LocFile	ConFile	SysFile	ExeSysFile	ProFile	ExeFile	File
SysAdmin	r,w,x (r)	r,w,x (r,w)	r,w,x (r,w)	r,w,x (r,w)	r,w,x	r,w,x	r,w,x	r,w,x
Mag	r	r,w	r,w		x	x	x	
OSDev	r	r,w	(r,w)	(r,w)	x (r,w,x)	x (r,w,x)	x (r,w,x)	
LocCli	r	r,w			x	x	x	
RemCli		r,w			x	x	x	

7. CONCLUSION AND OUTLOOK

We presented an approach for an access control model with object class hierarchy by means of DL. To sum up our results, we have

- fixed the flaws in the existing approach (Chae and Shiri, 2007)
- extended the idea of an object class hierarchy for RBAC by adding *actions* to the definition of a permission and gave a formal definition of RBAC-CH and applied the *concept product* in our OWL 1.1 ontology

The permissions are not defined explicitly for users to objects but for roles to object classes. Individual users are asserted to user roles and objects are asserted to object classes. The realization reasoning service makes object properties between ABox individuals explicit. For the RBAC-CH policy enforcement we need to run the reasoner only once and at runtime we only need to read out the inferred ABox to decide about authorization.

Our current work focuses on increasing usability: We want to adhere to the distinction of explicit and implicit permissions like we have compared them in the tables 2 and 3. We claim that this reduces effort to assign permissions and helps to avoid mistakes by granting unwanted permissions. We want to investigate two directions. In one direction, the user defines a complete matrix with the intended permissions, and the user role hierarchy and object hierarchy is then automatically derived to reduce the explicit permissions to a minimal set. This would help to get insights in the user role and object hierarchy which have not been obvious before. Methods from formal concept analysis (FCA) may be appropriate in this context. In the opposite direction, given a role hierarchy and object hierarchy, an access matrix can be completed with inferred permissions.

ACKNOWLEDGEMENTS

This research was 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. The responsibility for this publication lies with the authors.

REFERENCES

F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2. ed., 2007.

S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2 2004. W3C Recommendation, available at http://www.w3.org/TR/owl-ref/, retrieved January 3, 2008.

P. A. Bonatti and D. Olmedilla, "Rule-based policy representation and reasoning for the semantic web," in *Reasoning Web* (G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P.-L. Patranjan, and R. Tolksdorf, eds.), vol. 4636 of *Lecture Notes in Computer Science*, pp. 240–268, Springer, 2007.

J.-H. Chae and N. Shiri, "Formalization of RBAC policy with object class hierarchy," in *Information Security Practice and Experience (ISPEC)* (E. Dawson and D. S. Wong, eds.), vol. 4464 of *Lecture Notes in Computer Science*, pp. 162–176, Springer, 2007.

T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham, "ROWLBAC: representing role based access control in owl," in *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 73–82, ACM, 2008.

I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," in *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.

V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 677–686, ACM, 2007.

P. F. Patel-Schneider and I. Horrocks, *OWL 1.1 Web Ontology Language Overview*. World Wide Web Consortium, 12 2006. W3C Member Submission, available at http://www.w3.org/Submission/owl11-overview/, retrieved January 3, 2008.

S. Rudolph, M. Krötzsch, and P. Hitzler, "All elephants are bigger than all mice," in *Proceedings of the 21st International Workshop on Description Logics (DL2008)*, 2008.

R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role-based access control: towards a unified standard," in *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, (New York, NY, USA), pp. 47–63, ACM, 2000.

R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993.

C. Zhao, N. Heilili, S. Liu, and Z. Lin, "Representation and reasoning on RBAC: A description logic approach," in *Proceedings of Second International Colloquium Theoretical Aspects of Computing (ICTAC)*, pp. 381–393, 2005.