Using OWL DL Reasoning to decide about authorization in RBAC*

Martin Knechtel, Jan Hladik, and Frithjof Dau

SAP AG, SAP Research CEC Dresden, Germany
{martin.knechtel, jan.hladik, frithjof.dau}@sap.com

1 Introduction and Motivation

Role Based Access Control (RBAC) [1] is a standardized model to indirectly assign permissions to users by user roles. We follow the proposal of Chae and Shiri [2] to introduce a hierarchy of object classes in addition to the hierarchy of user roles along which permissions are inherited. This makes sense since e.g. in file systems the inheritance of permissions along the directory tree is common. Different formalizations are suitable for RBAC, especially Description Logics. Description Logic (DL) [3] systems provide their users with inference services that deduce implicit knowledge from the explicitly represented knowledge. The proposal by Chae and Shiri [2] is based on DL but has several flaws which we want to fix with this paper. The authors apply essential properties of DL in an incorrect way and do not respect DL semantics, do not use ABox assertions correctly, miss a discussion of the open world assumption and obtain wrong results with their running example. For a more detailed discussion of these issues, please refer to [4].

2 A DL Ontology for RBAC-CH

We decided to formally model RBAC-CH by means of a DL, and a crucial modeling issue is which DL constructors are requireed in order to decide which DL is used in our approach. On the side of the concept constructors, we essentially require only the constructors of $\mathcal{ALC}$, i.e. top and bottom, conjunction, disjunction, negation, and existential and value restrictions (we do not require number restrictions). Moreover, we use the "fills" constructor $P : a\ (= \exists P.\{a\})$ [3]. However, we do require some more powerful constructors for properties. Besides inverse properties and subproperty relationships, we require property chain inclusion axioms, i.e. axioms $P_1 \circ P_2 \sqsubseteq P$. This particular feature has been added to OWL in the step from OWL DL, which is based on the DL $\mathcal{SHOIN}(\mathbf{D})$, to OWL 2, which is based on $\mathcal{SROIQ}(\mathbf{D})$ [5]. For this reason, and as OWL 2 is likely to become the new standard, we decided to base our formalization on $\mathcal{SROIQ}(\mathbf{D})$.

* 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.

The ontology consists of ABox and TBox. The user role hierarchy, the object class hierarchy and permissions of user roles to object classes are defined in the TBox. The assertions of user individuals to user roles as well as object individuals to object classes are defined in the ABox. From these explicit facts a reasoner can infer the permissions of individual users to individual objects in the ABox by full calculation of property extensions, and they can be queried at runtime. Also permissions from the perspective of an object, called the Access Control List (ACL) can be queried.

Permissions are modeled by object properties. The definition of permissions for user roles on object classes has an important requirement: permissions are exhaustive for every individual of a user role and every individual of an object class, e.g. "all system administrators can read 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 [6] the *concept product* to express such statements, which can be captured in the very expressive DL $\mathcal{SROIQ}$. The concept product putting all individuals of concept C in relation to all individuals of concept D by object property P is written as $C \times D \sqsubseteq P$. The following steps are performed to simulate the concept product in $\mathcal{SROIQ}$: (1) delete the axiom $C \times D \sqsubseteq P$, (2) add a new generalized property chain inclusion axiom $P1 \circ P2 \sqsubseteq P$, where $P1, P2$ are fresh property names, (3) introduce fresh nominal $\{a\}$ and add TBox axioms $C \sqsubseteq \exists P1.\{a\}$ and $D \sqsubseteq \exists P2^{-}.\{a\}$. For further details refer to [6].

3 Working Example and Comparison

The TBox contains the concept hierarchy with the user roles as well as the object class hierarchy as depicted in Fig. 1 analogously to the scenario in [2]. 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).

Furthermore the TBox contains the assignment of permissions for user roles on objects. In the following example, we define that every remote client can execute every executable file with the concept product $RemCli \times ExeFile \sqsubseteq canExecute$. The concept product is simulated in ***SROIQ*** with $canBeExecutedBy \equiv canExecute^{-}$, $canExecute_1 \circ canExecute_2^{-} \sqsubseteq canExecute$, $RemCli \sqsubseteq \exists canExecute_1.\{a\}$, $ExeFile \sqsubseteq \exists canExecute_2.\{a\}$. The ABox contains assertions of users to user roles and files to object classes in the form $OSDev(edward)$, $ProFile(programFile_1)$ etc. After full calculation of property extensions 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 the Capability entry $canExecute(edward, programFile_1)$ and for the individual $programFile_1$ we find the ACL entry $canBeExecutedBy(programFile_1, edward)$. The resulting knowledge base is depicted in Fig. 2.

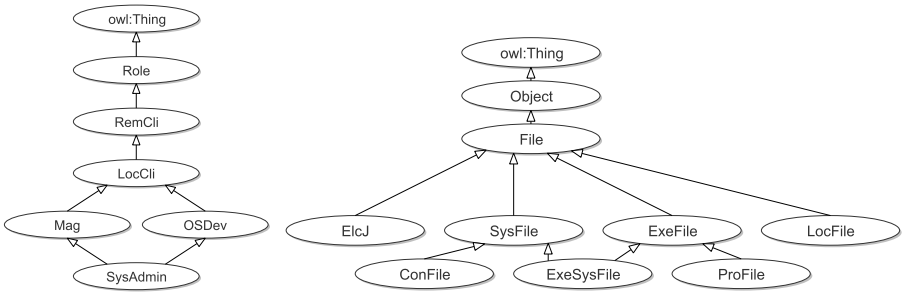

Fig. 1. User role and object class hierarchy (white arrow heads: SubClassOf relation)

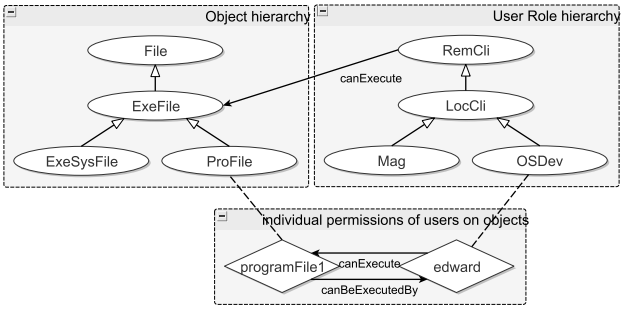

Fig. 2. Infer authorization of individuals from explicit permission assignments between concepts (oval: concept, diamond: individual, white arrow head: subsumption relation, black arrow head: object property, dashed line: concept assertion)

In the example we only defined "remote client can execute executable files". The example scenario from [2] we stick to contains further explicit permissions, which we have given in Tab. 1. We took them in our complete ontology. The definition is analogous 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 $Role \times ObjectClass \sqsubseteq action$.

Due to the user role hierarchy and the object class hierarchy, explicit permissions in Tab. 1 induce implicit permissions made explicit by the DL reasoner in Tab. 2. In [2] some inferences are not correct which we have additionally given in braces.

4 Conclusion and Outlook

We presented an approach for an access control model with object hierarchy by means of DL. We have fixed the flaws in the existing approach [2] and applied the *concept product* in our OWL 2 ontology. The permissions are not defined explicitly for users to objects but for user roles to object classes. Individual users are assigned to user roles

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

Table 1. Access Matrix with explicit permissions

	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**	

Table 2. Access Matrix with explicit and implied permissions caused by user role hierarchy and object class hierarchy (conflicting solution from [2] given in parentheses)

and objects are assigned to object classes. The full computation of property extensions makes permissions between individuals explicit.

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 1 and 2. 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 shall be able to define the intended permissions in the access matrix, and the user role hierarchy and object hierarchy is then automatically derived. 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 user role hierarchy and object class hierarchy, an access matrix can be completed and updated at changes with inferred permissions. This allows the user to keep track of inferred permission in addition to his explicit permissions.

References

1. 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.
2. 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.
3. 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.
4. M. Knechtel and J. Hladik, "RBAC authorization decision with DL reasoning," in *ICWI '08: Proceedings of the IADIS International Conference WWW/Internet*, 2008.
5. 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.
6. 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.