A Practical Framework for *RelBAC* Implementation

Lei Liu[1], Quanzhu Tao[1], Fausto Giunchiglia[2] and Rui Zhang[1⋆]

[1] CCST, Jilin University, Str. Qianjin 2699, Changchun, China 130012
liulei@jlu.edu.cn
tqzlimit@126.com
rui@jlu.edu.cn
[2] DISI, University of Trento, Italy
fausto@disi.unitn.it

Abstract. *RelBAC* is a new access control model that has gradually aroused the research interest in the domain of access control. But it is still not mature enough for industrial application due to its high logical complexity. In this paper, we present a framework to implement *RelBAC*. First, access control queries to *RelBAC* knowledge base (KB) are analysed and categorized into different queries as run-time or off-line. Then the necessary knowledge is studied to answer each type of query. We propose to separate the knowledge for run-time query, named as a complete ABox, from the classical *RelBAC* KB and store it in a relational database, so as to provide run-time answers within acceptable time. Last, a theorem is proved to backbone our method and an algorithm is proposed to calculate the complete ABox. This framework serves as a meaningful attempt to put *RelBAC* into practice.

Keywords: *RelBAC*, Description Logic, Complete ABox

1 Introduction

Access control models evolute with the advances of technologies. The famous Role-based Access Control model RBAC [4] was proposed in the 20th century, has now flourished in practical access control systems such as in Windows operation systems. It evolves into different models such as ARBAC [11], GeoRBAC [3], ARBAC [8] etc.

As a new access control model, *RelBAC* [6] connects the subject that query to access the object with the permission as a binary relation. The model provides intuitive and straightforward semantics to home users without formal access control experiences. But *RelBAC* has not flourished as expected in industry solutions. What hinders the model from practical application is the shortage of background supporting mechanism, which is supposed to be the powerful reasoning services provided by description logic (DL) reasoners. There is no good enough general purpose reasoners for *RelBAC* yet.

⋆ Corresponding Author

In this paper, we propose a practical framework to implement *RelBAC*. The knowledge base (KB) of a *RelBAC* access control system can be classified into different parts, namely, organization, authorization, constraint and environment. Each part consists of its typical structured axioms or assertions. Such structures are studied and classified into run-time and off-line queries to the KB. We prove that to answer the run-time query, only part of the ABox assertions are necessary. An algorithm is proposed to populate individuals of concepts to build a complete ABox. Such ABox assertions are separated from the classical KB and stored directly in a relational database. Then query from end user and/or system can be parsed and distributed to different engines, i.e. the database query engine to provide run-time response and the reasoning engine to check off-line queries. The framework provides a practical path to implement *RelBAC* in industrial solutions.

The paper is organized as follows. Section 2 gives a glance in *RelBAC* model; Section 3 illustrate our framework; Section 4 shows our strategy to implement *RelBAC*; and we conclude in Section 5.

2 Preliminaries

Relation-based Access Control (*RelBAC*) is an access control model introduced in [6] and formalized using the DL $\mathcal{ALCQIBO}$ as described in [16]. In this section, we will illustrate the basic *RelBAC* definitions and related access control policies.

2.1 Elementary

As shown in the ER Diagram of Figure 1, what distinguishes *RelBAC* from other access control models is the way it models permissions. A `PERMISSION` is modeled as an operation that users (`SUBJECT`s) can perform on certain resources (`OBJECT`s). To capture this intuition, a `PERMISSION` is named with the name of the operation it refers to. The *generalization* (loops) on each component repre-

Fig. 1. ER Diagram of *RelBAC*.

sents `IS-A` relations. Not only `SUBJECT` and `OBJECT` are organized along `IS-A` hierarchies but also `PERMISSION`.

2.2 Formalization

Together with *RelBAC*, a logic $\mathcal{ALCQIBO}$ extends the DL $\mathcal{ALC}$ [2] with qualified cardinalities, inverse roles, nominals and Boolean for roles (see [12, 10, 9] for

extensions of DLs with Booleans between roles). As described in [1], $\mathcal{ALCQIBO}$ is applied in access control domain to formalize *RelBAC*.

$\mathcal{ALCQIBO}$. The syntax of $\mathcal{ALCQIBO}$ is defined as follows.

$$\begin{aligned} C, D &::= A \mid \neg C \mid C \sqcap D \mid \geq n\, R.C \mid \{a_i\} \\ R, S &::= P \mid R^{-} \mid \neg R \mid R \sqcap S \end{aligned}$$

where $A \in \mathsf{N_C}$, $P \in \mathsf{N_R}$, $a_i \in \mathsf{N_I}$ and $n \in \mathbb{N}$.
A KB (KB) is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$, called *TBox*, is a finite set of *general concept inclusions (GCIs)* of the form $C \sqsubseteq D$ and a finite set of *general role inclusions (GRIs)* of the form $R \sqsubseteq S$, while $\mathcal{A}$, called *ABox*, is a finite set of concept and role assertions of the form $C(a_i)$ and $R(a_i, a_j)$, with $a_i, a_j \in \mathsf{N_I}$.

The corresponding semantics (partial) of $\mathcal{ALCQIBO}$ is defined as follows.

$$\begin{aligned} (R^{-})^{\mathcal{I}} &:= \{(y, x) \in \Delta \times \Delta \mid (x, y) \in R^{\mathcal{I}}\}, \\ (\neg R)^{\mathcal{I}} &:= \Delta \times \Delta \setminus R^{\mathcal{I}}, \quad (\neg C)^{\mathcal{I}} := \Delta \setminus C^{\mathcal{I}}, \\ (R \sqcap S)^{\mathcal{I}} &:= R^{\mathcal{I}} \cap S^{\mathcal{I}}, \quad (C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\ (\geqslant n\ R.C)^{\mathcal{I}} &:= \{x \in \Delta \mid \sharp\{y \in \Delta \mid (x, y) \in R^{\mathcal{I}} \text{ and } \\ &\qquad\qquad y \in C^{\mathcal{I}}\} \geq n\}, \\ \{a_i\}^{\mathcal{I}} &:= \{a_i^{\mathcal{I}}\}. \end{aligned}$$

An $\mathcal{ALCQIBO}$-interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is said to be a *model* of a KB, $\mathcal{K}$, iff it satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, for all $C \sqsubseteq D \in \mathcal{K}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, for all $R \sqsubseteq S \in \mathcal{K}$, $a_i^{\mathcal{I}} \in C^{\mathcal{I}}$, for all $C(a_i) \in \mathcal{A}$, and $(a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \in R^{\mathcal{I}}$, for all $R(a_i, a_j) \in \mathcal{A}$. In this case we say that $\mathcal{K}$ is satisfiable and write $\mathcal{I} \models \mathcal{K}$. A concept C (role R) is *satisfiable w.r.t.* $\mathcal{K}$ if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$ ($R^{\mathcal{I}} \neq \emptyset$).

Formal Specification. In *RelBAC* we distinguish five different kinds of specifications that, altogether, constitute an access control system: the organization information, the authorization policy, the control constraint, the environment factors and the administration query. *RelBAC* uses the description logic $\mathcal{ALCQIBO}$ to express each specification by associating a concept name to each `SUBJECT` and `OBJECT` while permissions are described by means of role names.

1. **Organization**, to organize the entities and relationships among entities into hierarchical structures with partial order.
2. **Authorization**, to declare the permissions from `SUBJECT` to `OBJECT`.
3. **Constraint**, to declare general regulations that existing and new authorization policies should follow.
4. **Query**, to check the instantiation or satisfiability of the current access control KB.

The first step of our work is to clarify the patterns latent in above specifications in order to analyze the difference of the reasoning services related to each pattern.

3 Framework

Based on the theory of *RelBAC*, we propose a framework towards the implementation of the theory. It consists of three layers coherent to a standard MVC structure, but specialized to fit the access control domain.

Figure 2 gives the conceptual model of the frame.

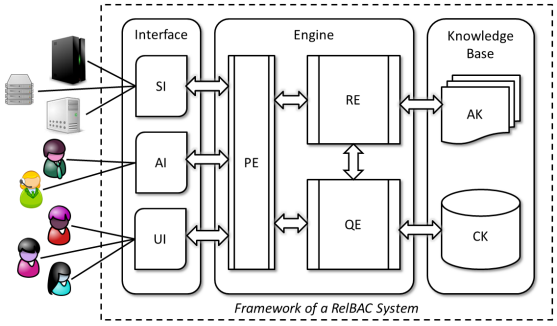

Fig. 2. Conceptual model of the frame for *RelBAC* Model.

As is shown in the figure, the framework consists of three major components, interface, engine and KB. Let us get into the details of them one by one.

Interface It is the intermediate between the system internal components and the outer 'users'. A channel, predefined or constructed at runtime, servers as interface is maintained by this component. It is not bounded to classical user interface but provides three sub components, SI, AI and UI.

- **SI** It stands for **S**ystem **I**nterface, which communicates with the outer information providers, such as time server, behavior monitor, audit record server, etc. Environment information are mainly transmitted to the system through SI.
- **AI** It stands for **A**dministrative **I**nterface, which connects administrators to the system via predefined graphical interfaces and facilitate tool interfaces.
- **UI** It stands for **U**sser **I**nterface, that provides classical graphical interfaces to end users. AI pages could be reused for UI, only with appropriate access control.

Engine It is the core part of the frame, that processes the info (updates, queries, maintenance, etc.) and communicate (if needed) with the KB. It mainly consists of three sub engines, PE, RE and QE.

- **PE** It is the **P**arser **E**ngine, which will preprocess the information from the interface, and forward it to appropriate engines for further processes. Different type of system queries will arrive, and fit in predefined or runtime studied patterns, then re-formulated into processable intermediate

format, thereafter be sent to responsible engines. Details of the formats and patterns will be discussed in Section 4.

RE It is the **R**easoning **E**ngine, that accepts OWL-API format reasoning tasks and provides reasoning services together with the interaction with the ontology KB(s).

QE It is the **Q**uery **E**ngine that takes the database queries as input and provides necessary query answers via access to the database in the KB.

KB It stores the knowledge necessary for the access control system functionality, such as the organization of the `SUBJECT`, `OBJECT` and `PERMISSION`; the authorization and constraint policies; the environment factors, etc. It is divided into two parts, the AK and DK, which stores the knowledge for different purposes.

AK It is the **A**dministrative **K**nowledge base, aims at general policies that support the administrative queries. It usually interacts with RE as the reasoning background.

CK It is the **C**ontrol **K**nowledge implemented via classical relational database techniques. Instantiated knowledge, such as the ABox assertion that 'John is permitted to update the root path' is stored here.

With the framework, the next step is to work out the details of each component. We will describe the theoretical details in the next section.

4 Implementation

This section will focus on the theoretical aspects to implement the framework introduced above. The patterns are clarified for the knowledge in a *RelBAC* system. Then the distribution strategy is proposed in details.

4.1 Knowledge Pattern

One of the key issue to implement the framework is to clarify the possible knowledge patterns. In Section 2.2, knowledge has been classified into four categories. Here, we will identify the patterns from the categories.

1. **Organization**: In the access control terminology, `SUBJECT`, `OBJECT` and `PERMISSION` may be organized in a taxonomy along the `IS-A` relation [5]. An `IS-A` relation is represented as a concept (or role) *inclusion* axiom in *RelBAC*:

$$C \sqsubseteq D \text{ or } P \sqsubseteq Q \tag{1}$$

where C, D are both `SUBJECT`s or both `OBJECT`s; P, Q are both `PERMISSION`s. `SUBJECT` and `OBJECT` are easy to be organized into `IS-A` hierarchies, especially with concerned attributes. Besides, compound concepts constructed with classical DL operator $\neg$ and $\sqcap$ may also exist in the formula above. The inversion operator $^-$ on role may also join in the *inclusion* axiom of roles in Formula (1).

Moreover, in addition to the DL roles for PERMISSION, classical roles may also exist to describe binary relations between entities that is not a PERMISSION, such as 'is-older-than' or 'has-published' relation in a system with an academic background. The *inverse role* operator

The number restriction operator $\geq$ is seldom used in organization. Its combination with the other operators will results in an axiom that could not be instantiated, as it puts restrictions on cardinality rather than fix individuals. This is more likely to be used in the specification of constraints.

2. **Authorization**: It specifies a permission existing between a SUBJECT and an OBJECT both on organization level or instance level in various of forms.

$$S \sqsubseteq \forall \neg P. \neg O \text{ or } O \sqsubseteq \forall \neg P^{-}. \neg S \tag{2}$$

which specifies that any SUBJECT in S is permitted to perform the operation (with the same name) P on any OBJECT in O. For easy reading purpose, it is transformed into a SWRL [14] rule as

$$S(?x), O(?y) \rightarrow P(?x, ?y) \tag{2'}$$

Special cases exist for Formula (2′), with alternation of the concept(s) with nominal(s), we have the following variations.

$$\begin{aligned} \{..., s_i, ...\} &\sqsubseteq \forall \neg P. \neg O \\ S &\sqsubseteq \forall \neg P. \neg \{..., o_j, ...\} \\ \{..., s_i, ...\} &\sqsubseteq \forall \neg P. \neg \{..., o_j, ...\} \end{aligned} \tag{2''}$$

where i, j are natural number indexes for individuals of SUBJECT and OBJECT. An authorization policy in the form of an ABox assertion $P(s, o)$ is only a special case of Formula (2″).

3. **Constraint**: It specifies the restrictions or regulations that the authorization policies should not violate. A constraint usually stays inactive in a running system. But when environment factors change, such as the rise of the system load, the crucial time point, the access behavior violation, etc. Therefore, the reasoning engine should check the consistency of the KB when new knowledge arrives or current knowledge updates.

   Some of the most concerned constraints are:

   (a) **Separation of Duties (SoD)**: It regulates the mutual exclusiveness of permissions. SoD lies in different levels and granularity. Given a set of positions $\mathcal{S} = \{S_1, \ldots, S_n\}$, where each S_i is a concept name, a SoD policy 'a subject can take all the positions in $\mathcal{S}$', may take the form of an unsatisfiable compound role (in contrast to an atomic role).

$$\bigsqcup_{i=1}^{C_n^m} (\bigsqcap_{j=1}^{m} S_{i_j}) \sqsubseteq \bot \tag{3}$$

where C_n^m is the binomial coefficient of 'n choose m'. A special case of Formula (3) is in condition of $m = n$, then the formula changes into

$$\prod_{j=1}^{n} S_j \sqsubseteq \bot \tag{3'}$$

.

(b) **Chinese Wall (CnW)**: The Chinese Wall property regulates conflict of interest (CoI), that 'the resources in the set of CoI could not be accessed by the same user'. Given a set of sensitive resources, $\mathcal{O} = \{O_1, ..., O_n\}$, and the corresponding operation $\mathcal{P} = \{P_1, ..., P_n\}$, a CnW policy may take the form as.

$$\prod_{i=1}^{n} \exists P_i.O_i \sqsubseteq \bot \tag{4}$$

Specifically, when the operation remains the same, say P, then the formula changes into

$$\geq 2\ P.(\bigsqcup_{i=1}^{n} O_i) \sqsubseteq \bot \tag{4'}$$

4. **Query**: A query searches for subjects, permissions and/or objects from the KB. A consistent KB is denoted as Σ hereafter. A query is classified as either control or administrative.

Control Query (CQ) It verifies wether a requester for the resource have or not the permission to access certain number of objects. Given a `SUBJECT` u, the query could be of the following patterns.
(a) whether a `SUBJECT` s has `PERMISSION` P on the `OBJECT` o;
(b) whether a `SUBJECT` s has `PERMISSION` P on the `OBJECT` in O.
correspond to the, so called, *instance checking* reasoning service:

$$\Sigma \models P(s, o), \tag{5}$$
$$\Sigma \models (\forall \neg P.\neg O)(s). \tag{6}$$

in which Formula (6) could be reformed into a satisfiability check as in Formula (8) and (9). We will discuss it later at the end of this subsection. It is obvious that all these three formulae of CQ should be answered within acceptable time by the system.

Administrative Query (AQ) It checks the state of the access control system, such as
(a) search for all the `SUBJECT` that has `PERMISSION` P on `OBJECT` o;
(b) search for all the `OBJECT` that is permitted to some `SUBJECT` s via `PERMISSION` P;
(c) whether the current system KB is consistent;
(d) whether an intended policy implied in the KB;
(e) whether an intended policy conflicts with the KB;
(f) whether an intended policy irrelevant to the KB;

(g) whether an intended update of environment violates the KB;
(h) whether an intended update consistent with the KB;
(i) whether an intended update irrelevant to the KB;
The first two AQ's lie in the reasoning of *instance retrieval* for a compound concept, say $\exists P.\{o\}$ and $\exists P^{-}.\{u\}$, as the following,

$$\text{retrieve all } \texttt{SUBJECT} \text{ u that} \Sigma \models \exists P.\{o\}(u) \tag{7}$$

$$\text{retrieve all } \texttt{OBJECT} \text{ o that} \Sigma \models \exists P^{-}.\{u\}(o) \tag{8}$$

Here the compound concept is not bounded to permission, environment attributes could be considered too. Therefore the concept on the right side of $\models$ could be *conjunction* or *union* of multiple concepts. The third AQ lies in *consistency checking* for the KB.

$$\Sigma \models \bot \tag{9}$$

If there is no model exists for Σ, the answer is *inconsistent*.
DL reasoning assumes an *Open World Assumption* [2], which does not imply a negation if the positive is not a deduction result, and vise versa. This means we cannot conclude that some policy (in format of an axiom) is unsatisfiable if it is not deduced from the KB. Therefore, given a concerned policy as an axiom A, the fourth AQ could be answered with a consistency checking as Formula (9), of the renewed KB as $\Sigma \models \{A\}$. The fifth AQ could be answered with a consistency checking of the renewed KB as $\Sigma \models \{\neg A\}$. The sixth AQ however, could not be answered with a single consistency check because of the open world assumption. It is answered as *irrelevant* only if the fourth and fifth queries are both answered *consistent*.
The last three AQ's are similar to AQ 4-6, with the only difference that instead of adding a new axiom A, an existing entity e, say a `SUBJECT`, `OBJECT` or `PERMISSION`, will be alternated by a 'new' entity with its attribute changed, i.e. all appearance of e will be replaced with an e'. Then the 'new' KB is checked in respect of Σ.

As mentioned for Formula (6), it could be decomposed into two steps.

1. Instance Retrieval: retrieve all the `OBJECT` o for the compound concept $\exists\ P^{-}.\{u\}$ with respect to Formula (8), which makes a set X;
2. Satisfiability Check: check the satisfiability of the axiom A in the form of $O \sqsubseteq X$ with respect to Formula (9).

Here in this subsection, nine patterns are discovered in Formulae (1 to 9). Different strategies will be applied on the patterns to enhance the run-time performance of the system.

4.2 Task Distribution

As is shown in Figure 2 in Section 3, the queries through interfaces are transformed by the Parse Engine PE and then transported to the other engines to

process via interaction(s) with the KB. However, a general purpose reasoner such as [15], [7] and [13] does not provide good enough responses to *RelBAC* queries. To be precise, they cannot reason on *RelBAC* with the logic $\mathcal{ALCQIBO}$., not to say provide run-time answers. How to transform *RelBAC* into a form that could be operated by DL reasoners will not be covered for page limits. The aim of this paper is to make the system work at run-time.

The strategy is to distribute the queries to different engines, i.e. RE or QE. With the pattern clarification of *RelBAC* in Section 4.1, we distinguish the knowledge patterns into nine formulae (Formula 1 - 9). We see that only the CQ's relate to run-time queries. Moreover, they are all related to instances. Therefore, we design an algorithm to populate all the individuals inside the KB into possible concepts which named as ABoxing, and get to a theorem as the following.

Definition 1. *The ABox assertion set $\mathcal{A}$ of an ontology $\mathcal{O}$ is complete, if and only if any ABox assertion α implied by $\mathcal{O}$ is explicitly inside $\mathcal{A}$.*

$$\mathcal{A} = \{\alpha | \alpha \text{ is an ABox assertion}, \mathcal{O} \models \alpha\}$$

Then we have the following theorem which gives that,

Theorem 1. *If the ABox assertion set of an ontology $\mathcal{A}$ is complete, then any answer by $\mathcal{O}$ to a CQ is the same as answered by $\mathcal{A}$.*

Proof. Given a CQ, the pattern of the query falls into one of the three form as Formula (5-6).

For a query in pattern of Formula (5), if $P(s, o)$ can be deduced from Σ then with Definition 1, it should be explicitly in $\mathcal{A}$, which will derive $P(s, o)$ apparently. If it is not deduced from Σ, either $\neg P(s, o)$ is deduced, or neither is deduced. For the first case, $\neg P(s, o)$ should be explicitly asserted in $\mathcal{A}$, therefore $P(s, o)$ cannot be derived; for the second case, without TBox axioms, $\mathcal{A}$ is only a set of assertions that could not deduce any fact that is not explicitly asserted in $\mathcal{A}$, then $P(s, o)$ cannot be derived.

For a query in pattern of Formula (6) if for each $o_i \in O$, $P(s, o_i)$ can be deduced from Σ, then s satisfies Formula (6); if $\mathcal{A}$ is complete then for each $o_i \in O$, $P(s, o_i)$ and $O(o_i)$ are explicitly asserted in $\mathcal{A}$, then can s is verified to fit Formula (6). Otherwise, if for all $o_i \in O$, there exists one $P(s, o_i)$ can not be deduced from Σ, it is then not asserted in $\mathcal{A}$, and cannot fit Formula (6).

An algorithm is proposed to populate individuals to concept in $\mathcal{A}$. In Algorithm 1, ABox assertion set and TBox axiom set are initialized on Line 1-2. The the loop in Line 3-10 computes all the necessary TBox axioms and explicitly adds them into $\mathcal{T}$. Then the second loop in Line 11-24 compute the ABox assertions according to the computed $\mathcal{T}$.

After population, $\mathcal{A}$ is complete. Each assertion in $\mathcal{A}$ could be coded into classical database record. Then the task to answer any CQ query is distributed to classical database queries. For the rest AQ, the task could be carried out with general purpose reasoners that support SWRL, which might be time-consuming, but acceptable for off-line administration.

Algorithm 1: ABox Population

Input: An ontology $\mathcal{O}$, consists of two parts namely the ABox assertions in a set $\mathcal{A}$ and the TBox axioms in a set $\mathcal{T}$.
Output: The complete ABox assertion set $\mathcal{A}$ of O

```
1  A ← O.A
2  T ← O.T
3  while T grows do
4      for each a ∈T do
5          if a is C ⊑ D then
6              T +={C ⊑ D}
7          if a is C ≡ D then
8              T +={C ⊑ D, D ⊑ C}
9          if a is C ⊓ D ⊑ E then
10             T +={C ⊑ E, D ⊑ E}
11 while A grows do
12     for each b ∈ A do
13         if b is C(x) then
14             for each a ∈ T do
15                 if a is C ⊑ D then
16                     A +={D(x)}
17         if b is R(x, y) then
18             for each a ∈ T do
19                 if a is R ⊑ S or R ≡ S then
20                     A +={S(x, y)}
21                 if a is ∀ R.⊤ ⊑ D then
22                     A +={D(x)}
23                 if a is ∀ R⁻.⊤ ⊑ D then
24                     A +={D(y)}
25 return A
```

5 Conclusion

RelBAC stands out of many other access control models for its rich expressiveness and formalism. Its logical complexity hinders its application in industry. We provide a framework to implement *RelBAC* in real-world software solutions. Queries that should be answered in run-time are distributed to classical database, that has complete ABox assertions coded into database records. Queries that could be answered off-line are distributed to DL reasoners with services of consistency checking, satisfiability checking, instance retrieval etc. With this combination in our framework, advantages of each query-answer mechanism could be taken and a practical implementation of *RelBAC* is foreseen.

References

1. Artale, A., Crispo, B., Giunchiglia, F., Turkmen, F., Zhang, R.: Reasoning about relation based access control. In: Xiang, Y., Samarati, P., Hu, J., Zhou, W., Sadeghi, A.R. (eds.) NSS. pp. 231–238. IEEE Computer Society (2010)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York, NY, USA (2003)
3. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: Geo-rbac: A spatially aware rbac. ACM Transactions on Information and System Security 10(1) (2007)
4. Ferraiolo, D.F., Cugini, J.A., Kuhn, D.R.: Role-Based Access Control (RBAC): Features and Motivations. In: 11th Annual Computer Security Applications Proceedings (1995)
5. Giunchiglia, F., Zaihrayeu, I.: Lightweight ontologies. In: Encyclopedia of Database Systems. Springer (2008)
6. Giunchiglia, F., Zhang, R., Crispo, B.: Relbac: Relation based access control. In: SKG '08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid. pp. 3–11. IEEE Computer Society, Washington, DC, USA (2008)
7. Haarslev, V., Möller, R.: Racer: A core inference engine for the semantic web. In: Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20. pp. 27–36 (2003)
8. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. IEEE Computer 43(6), 79–81 (2010), http://dblp.uni-trier.de/db/journals/computer/computer43.html#KuhnCW10
9. Lutz, C., Sattler, U.: The complexity of reasoning with boolean modal logics. In: Wolter, F., Wansing, H., de Rijke, M., Zakharyaschev, M. (eds.) Advances in Modal Logics Volume 3. CSLI Publications, Stanford (2001)
10. Lutz, C., Walther, D.: Pdl with negation of atomic programs. Journal of Applied Non-Classical Logic 15(2), 189–214 (2005)
11. Oh, S., Sandhu, R.S.: A model for role administration using organization structure. In: SACMAT. pp. 155–162 (2002)
12. Schmidt, R.A., Tishkovsky, D.: Using tableau to decide expressive description logics with role negation. In: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC + ASWC. Lecture Notes in Computer Science, vol. 4825, pp. 438–451 (2007)
13. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Submitted for publication to Journal of Web Semantics. (2003)
14. SWRL: Http://www.w3.org/Submission/SWRL/
15. Tsarkov, D., Horrocks, I.: Fact++ description logic reasoner: System description. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006). Lecture Notes in Artificial Intelligence, vol. 4130, pp. 292–297. Springer (2006)
16. Zhang, R., Artale, A., Giunchiglia, F., Crispo, B.: Using description logics in relation based access control. In: Proceedings of the DL Home 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009 (2009)